循序渐进

Vue.js 3

前端开发实战

• 张益珲 编著 •

清華大學出版社
北京

内 容 简 介

本书以一个多年前端“老司机”的视角，循序渐进地介绍当前流行的前端框架 Vue.js 3 的新特性、各项功能及其在商业开发中的应用。全书共 15 章，第 1~6 章介绍 Vue.js 3 的模板、组件、交互处理等基础知识；第 7 章介绍 Vue.js 3 框架的响应式原理及组合式 API；第 8 章介绍使用 Vue.js 3 框架开发前端动画效果；第 9 章介绍开发大型项目必备的脚手架工具 VueCli 和 Vite；第 10 章介绍基于 Vue.js 3 的 UI 框架 Element Plus；第 11~13 章分别介绍网络请求框架 vue-axios、路由管理框架 Vue Router、状态管理框架 Vuex；第 14 章和第 15 章介绍两个相对完整的项目的开发，即学习网站和电商后台系统。本书试图介绍 Vue.js 3 全家桶及周边框架和工具的综合应用，旨在使读者通过阅读本书开发自己的应用程序。本书还在各章安排了小型范例和练习题，并提供了教学视频、源代码及 PPT 课件。

本书既可以入门，也可以进阶，适合 Vue.js 3 初学者和前端开发人员使用，也可以作为网课、培训机构与大中专院校的教学用书。

图书在版编目（CIP）数据

循序渐进 Vue.js 3 前端开发实战 / 张益珲编著.—北京：清华大学出版社，2021.12
ISBN 978-7-302-59565-6

Ⅰ.①循… Ⅱ.①张… Ⅲ.①网页制作工具－程序设计 Ⅳ.①TP393.092.2

中国版本图书馆 CIP 数据核字（2021）第 232842 号

责任编辑：王金柱
封面设计：王 翔
责任校对：闫秀华
责任印制：

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-62770175 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：
经 销：全国新华书店
开 本：190mm×260mm **印 张：**21 **字 数：**538 千字
版 次：2022 年 1 月第 1 版 **印 次：**2022 年 1 月第 1 次印刷
定 价：79.80 元

产品编号：091192-01

前　言

当下互联网已经成为人们生活中必不可少的一部分，无论是在互联网上查询信息、玩游戏，还是通过互联网购物、付款，都离不开成千上万的互联网应用程序的支持。不知你有没有想过，这些互联网应用是如何开发出来的呢？开发一个外观精美、功能强大的网站需要多长时间？我们是否具备这样的能力来将自己的想法、创意开发成产品？如果你曾有过类似的思考，那么本书可以帮你快速掌握这一技能。

阅读任何一本图书的过程本身就是一个学习的过程，扩展自己的知识边界、提高自己的技能水平可能并非一件容易的事，但却是一件有趣的事。如果你能够自主开发功能完备的商业项目，能够通过自己开发的应用程序为更多的人带来生活上的帮助，这一定是一件非常令人快乐的事情。因此，若要品尝幸福的果实，一定要经历开花结果过程中的辛劳，本书会尽量帮助你将这一过程缩短。

突出应用开发的内容安排

在章节安排上，本书共分15章。从前端基础讲起，深入浅出地介绍了Vue.js框架的功能用法及实现原理。并且，几乎每一章的最后都安排了实践与练习，力求能使读者边学边练，快速且扎实地掌握Vue.js框架的方方面面，并且可以真正使用它开发出商业级别的应用程序。

第 1 章是本书的入门章节，简单介绍了前端开发必备的基础知识，包括 HTML、CSS 和 JavaScript 这 3 种前端开发必备的技能。这些虽然不是本书的重点，但却是学习 Vue 前必须掌握的基础技能，如果读者在阅读本章时对书中所介绍的技术不甚了解，建议还是先不要阅读后续章节，毕竟没有基础，应用将无从谈起。

第 2 章介绍 Vue 模板的基本用法，包括模板插值、条件与循环渲染的相关语法。这些功能是 Vue.js 框架提供的基础功能，使用它们能使我们在开发网页应用时事半功倍。

第3章介绍了Vue组件中属性和方法的相关概念，将使用面向对象的思路来进行前端程序开发，本章的最后会通过一个功能简单的登录注册页面来对读者的掌握情况进行检验。

第4章将介绍前端应用中用户交互的处理方法，一个网页如果不能进行用户交互，那么将如一潭死水，毫无生机，用户交互为应用程序带来灵魂。本章除了介绍基础的网页用户交互的处理外，还将结合Vue.js来讲解如何在Vue.js框架中更加高效地处理用户交互事件。

第 5 章和第 6 章将由浅入深地介绍 Vue.js 中组件的相关应用。组件是 Vue.js 框架的核心，在实际的应用开发中，更是离不开自定义组件技术。有了组件，我们才有了开发大型互联网应用的基础，组件使得项目的结构更加便于管理，工程的可维护与可扩展性大大提高，且组件本身的复用性也使开发者可以大量使用第三方模块或将自己开发的模块作为组件供各种项目使用，极大地提高了开发效率。

第7章介绍Vue.js框架的响应式原理，以及Vue.js 3.0版本引入的组合式API新特性。本章是对读

者开发能力的一种拔高，引导读者从实现功能到精致逻辑设计的方向进步。

第 8 章将介绍通过 Vue.js 框架方便地开发前端动画效果。动画技术在前端开发中也非常重要，前端是直接和用户面对面的，功能本身只是前端应用的一部分，更重要的是给用户带来良好的体验，合理地使用动画是提升用户体验的一大法宝。

第9章会给读者介绍开发大型项目必备的脚手架VueCli的基本用法，管理项目、编译打包项目都需要使用此脚手架工具；本章还介绍了Vue CLI与新一代前端构建工具Vite的优缺点以及Vite的简单使用。

通过前9章的学习，对于Vue.js框架本身的功能与用法基本学习完毕。但是对于开发一款商业级的互联网应用，这还远远不够，第10章将介绍样式美观且扩展性极强的基于Vue.js的UI框架Element Plus；第11章将网络请求框架vue-axios；第12章将介绍一款非常好用的Vue应用路由管理框架Vue Router；第13章会介绍强大的状态管理框架Vuex，使用它开发者可以更好地管理大型Vue项目各个模块间的交互。

第14章和第15章将通过两个相对完整的应用项目来全面地对本书涉及的Vue.js技能进行综合应用，帮助读者学以致用，更加深入地理解所学习的内容。

实用丰富的配套资源

为使读者能够轻松地掌握本书内容，本书录制了配套的视频教程，对于初学者来说，通过观看教学视频，可以大幅降低学习难度，提升学习效率，可以说是物超所值。读者扫描本书的二维码即可观看。

本书还提供了源代码和PPT课件，以方便读者上机演练和巩固学习成果，读者扫描以下二维码即可下载：

如果你在下载过程中遇到问题，可发送邮件至booksaga@126.com获得帮助，邮件标题为“循序渐进Vue.js 3前端开发实战”。

最后，对于本书的出版，要感谢支持我的家人和朋友，感谢清华大学出版社王金柱编辑的辛勤付出。在王编辑的指导下，才能完成本书的选题策划、章节规划、内容修正等工作。重中之重是，感谢读者们的耐心，由衷地希望本书可以带给你预期的收获。无论是学习还是工作，都希望你在阅读本书后能够更上一层楼。同时，由于时间仓促和能力所限，书中难免存在疏漏之处，希望读者不吝指教。

编　者

2021 年 10 月 18 日　上海

目　录

第1章

从前端基础到 Vue.js 3

前端技术是互联网大技术栈中非常重要的一个分支。前端技术本身也是互联网技术发展的见证，其就像一扇窗户，展现了互联网技术的发展与变迁。

前端技术通常是指通过浏览器将信息展现给用户这一过程中涉及的互联网技术。随着目前前端设备的泛化，并非所有的前端产品都是通过浏览器来呈现的，例如微信小程序、支付宝小程序、移动端应用等被统称为前端应用，相应地，前端技术栈也越来越宽广。

讲到前端技术，虽然目前有各种各样的框架与解决方案，基础的技术依然是前端三剑客：HTML5、CSS3与JavaScript。随着HTML5与CSS3的应用，前端网页的美观程度与交互能力都得到了很大的提升。

本章作为准备章节，向读者简单介绍前端技术的发展过程，以及前端三剑客的基本概念与应用，并简单介绍响应式开发框架的相关概念。本章还将通过一个简单的静态页面来向读者展示如何使用HTML、CSS与JavaScript代码将网页展示到浏览器界面中。

通过本章，你将学习到：

- 了解前端技术的发展概况。
- 对 HTML 技术有简单的了解。
- 对 CSS 技术有简单的了解。
- 对 JavaScript 技术有简单的了解。
- 认识渐进式界面开发框架 Vue，初步体验 Vue 开发框架。

1.1 前端技术演进

说起前端技术的发展历程，还是要从HTML说起。1990年12月，计算机学家Tim Berners-Lee使用HTML语言在NeXT计算机上部署了第一套由“主机-网站-浏览器”构成的Web系统，我们通常认为这是世界上第一套完整的前后端应用，将其作为Web技术开发的开端。

1993年，第一款正式的浏览器Mosaic发布，1994年年底，W3C组织成立，标志着互联网进入了标准化发展的阶段，互联网技术迎来快速发展的春天。

1995年，网景公司推出JavaScript语言，赋予了浏览器更强大的页面渲染与交互能力，使之前的静态网页开始真正地向动态化的方向发展，由此后端程序的复杂度大幅度提升，MVC开发架构诞生，其中前端负责MVC架构中的V（视图层）的开发。

2004年，Ajax技术在Web开发中得到应用，使得网页可以灵活地使用HTTP异步请求来动态地更新页面，复杂的渲染逻辑由之前的后端处理逐渐更替为前端处理，开启了Web 2.0时代。由此，类似jQuery等非常多流行的前端DOM处理框架相继诞生，以其中最流行的jQuery框架为例，其几乎成为网站开发的标配。

2008年，HTML5草案发布，2014年10月，W3C正式发布HTML5推荐标准，众多流行的浏览器也都对其进行了支持，前端网页的交互能力大幅度提高。前端网站开始由Web Site向Web App进化，2010年开始相继出现了Angular JS、Vue JS等开发框架。这些框架的应用开启了互联网网站开发的SPA时代，即单页面应用程序时代（Single Page Application），这也是当今互联网Web应用开发的主流方向。

总体来说，前端技术的发展经历了静态页面阶段、Ajax阶段、MVC阶段，最终发展到了SPA阶段。

在静态页面阶段，前端代码只是后端代码中的一部分，浏览器中展示给用户的页面都是静态的，这些页面的所有前端代码和数据都是后端组装完成后发送给浏览器进行展示的，页面响应速度慢，只能处理简单的用户交互，样式也不够美观。

在Ajax阶段，前端与后端实现了部分分离。前端的工作不再仅仅是展示页面，还需要进行数据的管理与用户的交互。当前端发展到Ajax阶段时，后端更多的工作是提供数据，前端代码逐渐变得复杂。

随着前端要完成的功能越来越复杂，代码量也越来越大。应运而生的很多框架都为前端代码工程结构管理提供了帮助，这些框架大多采用MVC或MVVM模式，将前端逻辑中的数据模型、视图展示和业务逻辑区分开来，为更高复杂性的前端工程提供了支持。

前端技术发展到SPA阶段则意味着网站不再单单用来展示数据，其是一个完整的应用程序，浏览器只需要加载一次网页，用户即可在其中完整使用多页面交互的复杂应用程序，程序的响应速度快，用户体验也非常好。

1.2　HTML 入门

首先，HTML是一种编程语言，是一种描述性的网页编程语言。HTML的全称为Hyper Text Markup Language，我们通常也将其称为超文本标记语言，所谓超文本，是指其除了可以用来描述文本信息外，还可以描述超出基础文本范围的图片、音频、视频等信息。

虽然说HTML是一种编程语言，但是从编程语言的特性上来看，HTML并不是一种完整的编程语言，其并没有很强的逻辑处理能力，更确切的说法为HTML是一种标记语言，其定义了一套标记标签用来描述和控制网站的渲染。

标签是HTML语言中非常重要的一部分，标签是指由尖括号包围的关键词，例如<h1>、<html>等。在HTML文档中，大多数标签都是成对出现的，例如<h1></h1>，在一对标签中，前面的标签是开始标签，后面的标签是结束标签。例如下面就是一个非常简单的HMTL文档示例：

```
<html>
<body>
<h1>Hello World</h1>
<p>HelloWorld 网页</p>
</body>
</html>
```

上面的代码中共有4对标签：html、body、h1和p，这些标签的排布与嵌套定义了完整的HTML文档，最终会由浏览器进行解析渲染。

1.2.1　准备开发工具

HTML文档本身也是一种文本，我们可以使用任何文本编辑器进行HTML文档的编写，只需使用.html文本后缀名即可。但是使用一个强大的HTML编辑器可以极大地提高编写效率，例如很多HTML编辑器都会提供代码提示、标签高亮、标签自动闭合等功能，这些功能都可以帮助我们在开发中十分快速地编写代码，并且可以减少因为笔误所产生的错误。

Visual Studio Code（VSCode）是一款非常强大的编辑器，其除了提供语法检查、格式整理、代码高亮等基础编程常用的功能外，还支持对代码进行调试和运行以及版本管理。通过安装扩展，VSCode几乎可以支持目前所有流行的编程语言。本书示例代码的编写也将采用VSCode编辑器完成。你可以在官方网站（https://code.visualstudio.com）下载新的VSCode编辑器。

目前，VSCode支持的操作系统有macOS、Windows和Linux，在网站中下载适合自己操作系统的VSCode版本进行安装即可，如图1-1所示。

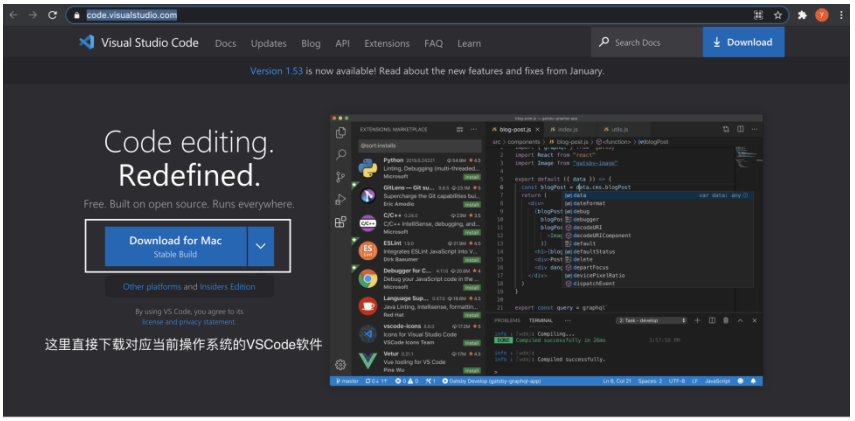


图 1-1　下载 VSCode 编辑器软件

下载安装VSCode软件后，我们可以尝试使用其创建一个简单的HTML文档，新建一个名为test.html的文件，在其中编写如下测试代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>HelloWorld</h1>
</body>
</html>
```

相信在输入代码的过程中，你已经能够体验到使用VSCode编程带来的畅快体验，并且在编辑器中关键词的高亮显示和自动缩进也使代码结构看上去更加直观，如图1-2所示。

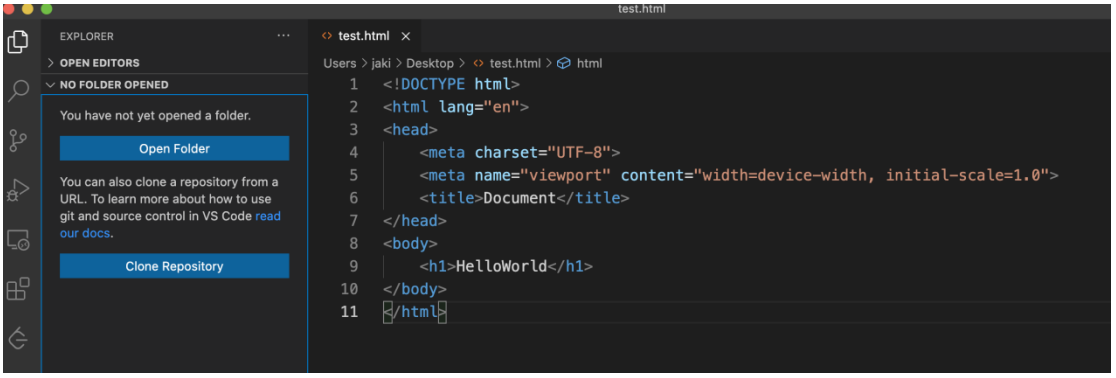


图 1-2　VSCode 的代码高亮显示与自动缩进功能

在VSCode中将代码编写完成后，我们可以直接对其进行运行。对于HTML源文件的运行，VSCode会自动将其以浏览器的方式打开，选择VSCode工具栏中的Run→Run Without Debugging选项，如图1-3所示。

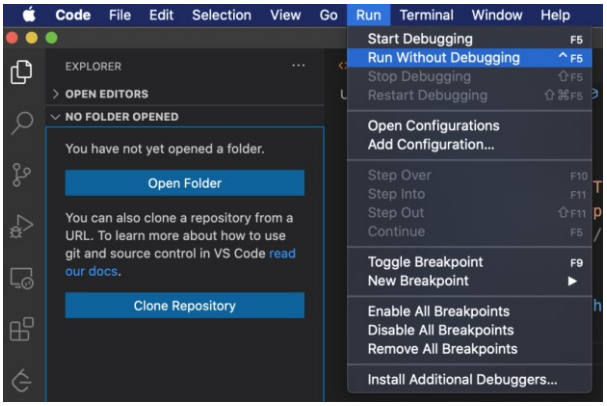


图 1-3　运行 HTML 文件

之后会弹出环境选择菜单，我们可以选择一款浏览器进行预览，如图1-4所示。建议安装Google Chrome浏览器，其有很多强大的插件可以帮助我们进行Web程序的调试。

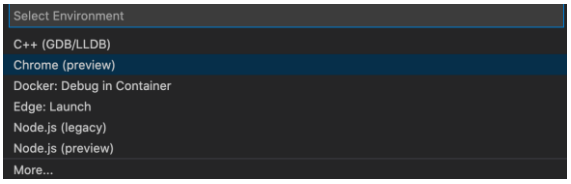


图 1-4　使用浏览器进行预览

预览效果如图1-5所示。

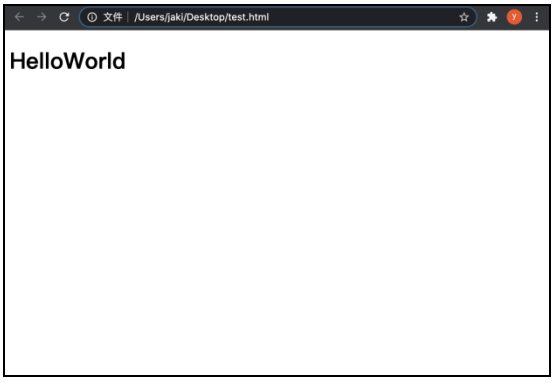


图 1-5　使用 HTML 实现的 HelloWorld 程序

1.2.2　HTML 中的基础标签

HTML中预定义了很多标签，本节通过几个基础标签的应用实例来向读者介绍标签在HTML中的简单用法。

HTML文档中的标题通常使用h标签来定义，根据标题的等级h标签又分为h1~h6共6个等级。使用VSCode编辑器创建一个名为base.html的文件，在其中编写如下代码：

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>基础标签应用</title>
</head>
<body>
    <h1>1级标题</h1>
    <h2>2级标题</h2>
    <h3>3级标题</h3>
    <h4>4级标题</h4>
    <h5>5级标题</h5>
    <h6>6级标题</h6>
</body>
</html>
```

我们后面的大多示例HTML文档的基本格式都是一样的，代码的不同之处主要在body标签内，后面的示例只会展示核心body中的代码。

运行上面的HTML文件，浏览器渲染效果如图1-6所示。可以发现，不同等级的标题文本字体的字号大小是不同的。

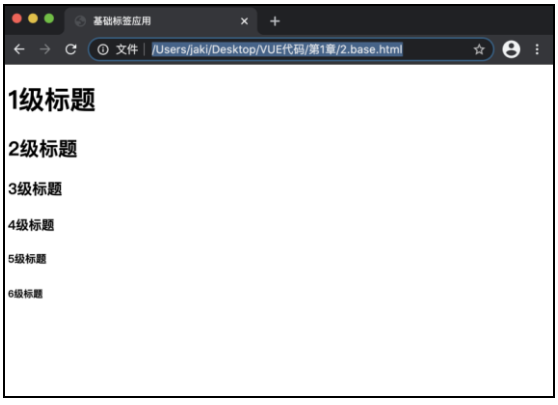


图 1-6　HTML 中的 h 标签

HTML文档的正文部分通常使用p标签定义，p标签的意义是段落，正文中的每个段落的文本都可以将其包裹在p标签内，如下：

```
    <p>这里是一个段落</p>
    <p>这里是一个段落</p>
```

a标签用来定义超链接，a标签中的href属性可以指向一个新的文档路径，当用户单击超链接的时候，浏览器会跳转到超链接指向的新网页，例如：

```
<a href="https://www.baidu.com">跳转到百度</a>
```

在实际的应用开发中，我们将很少使用a标签来处理网页的跳转逻辑，更多时候使用JavaScript来操作跳转逻辑。

HTML文档中也可以方便地显示图像，我们向base.html文件所在的目录中添加一张图片素材

（demo.png），使用img标签来定义图像，如下：

```
<div><img src="demo.png" alt="图片" width="400px"></div>
```

需要注意，之所以将img标签包裹在div标签中，是因为img标签是一个行内元素，如果我们想让图片单独另起一行展示，则需要使用div标签包裹，示例效果如图1-7所示。

图 1-7　HTML 文档效果演示

HTML中的标签可以通过属性对其渲染或交互行为进行控制，例如上面的a标签，href就是一种属性，其用来定义超链接的地址。在img标签中，src属性定义图片素材的地址，width属性定义图片渲染的宽度。标签中的属性使用如下格式设置：

```
tagName = "value"
```

tagName为属性的名字，不同的标签支持的属性也不同。通过设置属性，我们可以方便地对HTML文档中元素的布局与渲染进行控制，例如对于h1标签来说，将其align属性设置为center后，其就会在文档中居中展示：

```
<h1 align = "center">1级标题</h1>
```

效果如图1-8所示。

图 1-8　标题居中展示

HTML中还定义了一种非常特殊的标签：注释标签。编程工作除了要进行代码的编写外，优雅地撰写注释也是非常重要的。注释的内容在代码中可见，但是对浏览器来说是透明的，不会对渲染产生任何影响，示例如下：

```
<!-- 这里是注释的内容 -->
```

1.3 CSS 入门

通过1.2节的介绍，我们了解了HTML文档通过标签来进行框架的搭建和布局，虽然通过标签的一些属性也可以对展示的样式进行控制，但是其能力非常有限，我们在日常生活中看到的网页往往是五彩斑斓、多姿多彩的，这都要归功于CSS的强大能力。

CSS的全称是Cascading Style Sheets，即层叠样式表。其用处是定义如何展示HTML元素，通过CSS来控制网页元素的样式极大地提高了编码效率，在实际编程开发中，我们可以先将HTML文档的整体框架使用标签定义出来，之后使用CSS来对样式细节进行调整。

1.3.1 CSS 选择器入门

CSS代码的语法规则主要由两部分构成：选择器和声明语句。

声明语句用来定义样式，而选择器则用来指定要使用当前样式的HTML元素。在CSS中，基本的选择器有通用选择器、标签选择器、类选择器和id选择器。

1. 通用选择器

使用“*”来定义通用选择器，通用选择器的意义是对所有元素生效。创建一个名为selector.html的文件，在其中编写如下示例代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS选择器</title>
    <style>
        * {
            font-size: 18px;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <h1>这里是标题</h1>
    <p>这里是段落</p>
    <a>这里是超链接</a>
</body>
```

```
</html>
```

运行代码，浏览器渲染效果如图1-9所示。

图 1-9　HTML 渲染效果

如以上代码所示，使用通用选择器将HTML文档中的所有元素选中，之后将其内所有的文本字体都设置为粗体18号。

2. 标签选择器

标签选择器，顾名思义，我们可以通过标签名对此标签对应的所有元素的样式进行设置。示例代码如下：

```
p {
    color:red;
}
```

上面的代码将所有p标签内部的文本颜色设置为红色。

3. 类选择器

类选择器需要集合标签的class属性进行使用，我们可以在标签中添加class属性来为其设置一个类名，类选择器会将所有设置对应类名的元素选中，类选择器的使用格式为“.className”。

4. id 选择器

id选择器和类选择器类似，id选择器会通过标签的id属性进行选择，其使用格式为“#idName”。

示例如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS选择器</title>
```

```
        <style>
            * {
                font-size: 18px;
                font-weight: bold;
            }
            p {
                color:red;
            }
            .p2 {
                color: green;
            }
            #p3 {
                color:blue;
            }
        </style>
    </head>
    <body>
        <h1>这里是标题</h1>
        <p>这里是段落一</p>
        <p class="p2">这里是段落二</p>
        <p id="p3">这里是段落三</p>
        <a>这里是超链接</a>
    </body>
    </html>
```

运行上面的代码，可以看到“段落一”的文本被渲染成了红色，“段落二”的文本被渲染成了绿色，“段落三”的文本被渲染成了蓝色。

除了上面列举的4种CSS基本选择器外，CSS选择器还支持组合和嵌套，例如我们要选中如下代码中的p：

```
<div><p>div中嵌套的p</p></div>
```

可以使用后代选择器：

```
div p {
   color: cyan;
}
```

对于要同时选中多种元素的场景，我们也可以将各种选择器进行组合，每种选择器间使用逗号分隔即可，例如：

```
.p2, #p3 {
    font-style: italic;
}
```

此外，CSS选择器还有属性选择器、伪类选择器等，有兴趣的读者可以在互联网上查到大量的相关资料进行学习。本节只需要掌握基础的选择器使用方法即可。

1.3.2　CSS 样式入门

掌握了CSS选择器的应用，我们要选中HTML文档中的任何元素都非常容易，在实际开发中，常用的选择器是类选择器，我们可以根据组件的不同样式将其定义为不同的类，通过类选择器来对组件进行样式定义。

CSS提供了非常丰富的样式供开发者进行配置，包括元素背景的样式、文本的样式、边框与边距样式、渲染的位置等。本节将为读者介绍一些常用的样式配置方法。

1. 元素的背景配置

在CSS中，与元素的背景配置相关的属性都是以background开头的。使用CSS对元素的背景样式进行设置，可以实现相当复杂的元素渲染效果。常用的背景配置属性如表1-1所示。

表 1-1　CSS 背景属性配置

属 性 名	意　义	可配置值
background-color	设置元素背景颜色	这个属性可以接收任意合法的颜色值
background-image	设置元素的背景图片	图片素材的 URL
background-repeat	设置背景图片的填充方式	repeat-x：水平方向上重复 repeat-y：垂直方向上重复 no-repeat：图片背景不进行重复平铺
background-position	设置图片背景的定位方式	可以设置为相关定位的枚举值，如 top、center 等，也可以设置为长度值

2. 元素的文本配置

元素的文本配置包括对齐方式配置、缩进配置、文字间隔配置等。下面的CSS代码将演示这些文本配置属性的使用方式。

HTML标签：

```
<div class="text">文本配置属性 HelloWorld</div>
```

CSS设置：

```
.text {
    text-indent: 100px;
    text-align: right;
    word-spacing: 20px;
    letter-spacing: 10px;
    text-transform: uppercase;
    text-decoration: underline;
}
```

效果如图1-10所示。

图 1-10　使用 CSS 对文本元素进行配置

3. 边框与边距配置

使用CSS可以对元素的边框进行设置，例如设置元素的边框样式、宽度、颜色等。示例代码如下：

HTML元素：

```
<div class="border">设置元素的边框</div>
```

CSS设置：

```
.border {
    border-style: solid;
    border-width: 4px;
    border-color: red;
}
```

上面的示例代码中，border-style属性用于设置边框的样式，例如solid将其设置为实线；border-width属性用于设置边框的宽度，border-color属性用于设置边框的颜色。上面的代码运行后的效果如图1-11所示。

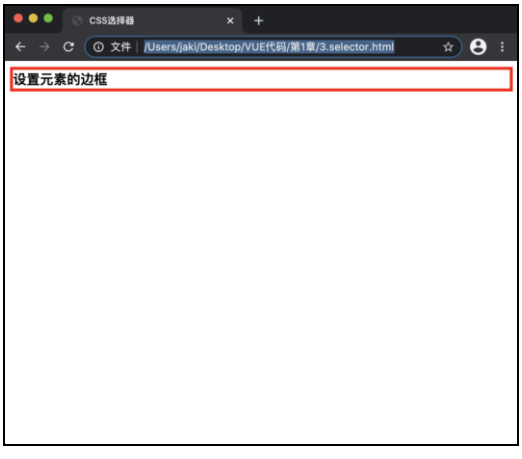


图 1-11　边框设置效果

使用border开头的属性配置会默认对元素的4个边框都进行设置，也可以单独对元素某个方向的边框进行配置，使用border-left、border-right、border-top、border-bottom开头的属性进行设置即可。

元素定位是CSS非常重要的功能之一，我们看到的网页之所以多姿多彩，都要归功于CSS可以灵活地对元素进行定位。

在网页布局中，CSS盒模型是一个非常重要的概念，其通过内外边距来控制元素间的相对位置。盒模型结构如图1-12所示。

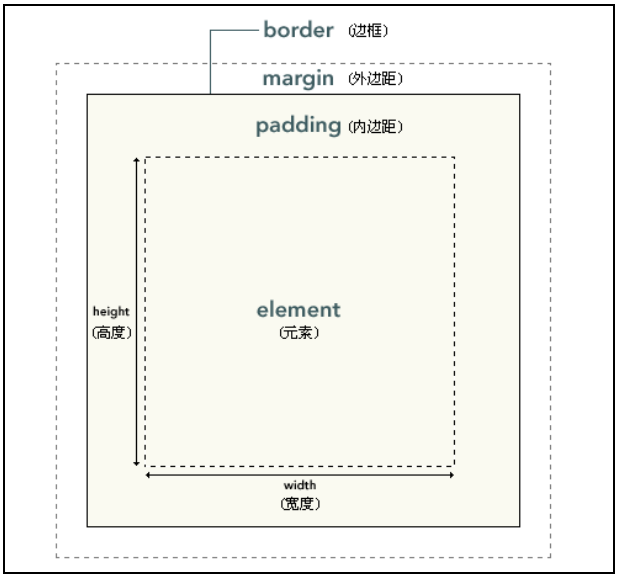


图 1-12　CSS 盒模型示意图

可以通过CSS的height和width属性控制元素的宽度和高度，padding相关的属性可以设置元素的内边距，可以使用padding-left、padding-right、padding-top和padding-bottom控制4个方向上的内边距。margin相关的属性用来控制元素的外边距，使用margin-left、margin-right、margin-top和margin-bottom控制4个方向的外边距。通过margin和padding的设置，可以灵活地控制元素间的相对位置。示例如下：

HTML元素：

```
<span class="sp1">sp1</span>
<span class="sp2">sp2</span>
<span class="sp3">sp3</span>
<span class="sp4">sp4</span>
```

CSS设置：

```
.sp1 {
   background-color: red;
   color: white;
   padding-right: 30px;
}
.sp2 {
```

```
        background-color: blue;
        color: white;
        padding-left: 30px;
    }
    .sp3 {
        background-color: green;
        color: white;
        margin-left: 30px;
    }
    .sp4 {
        background-color: indigo;
        color: white;
        margin-right: 30px;
    }
```

页面渲染效果如图1-13所示。

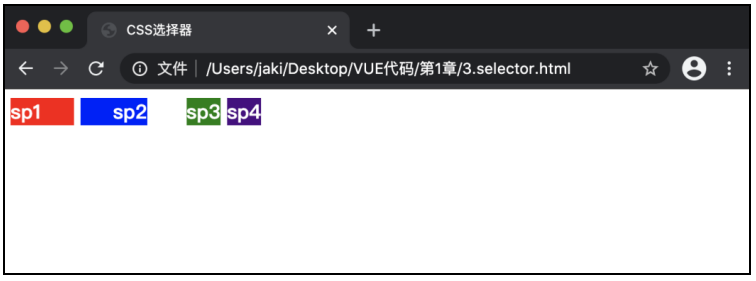


图 1-13　控制元素内外边距

需要注意，上面的元素之所以在一行展示，是因为span标签定义的元素默认为行内元素，不会自动换行布局。

关于元素的绝对定位与浮动的相关内容，不作为我们要了解的重点，在本书后续的测试案例中，我们会逐步使用这些技术为读者演示。

1.4　JavaScript 入门

学习Vue开发技术，JavaScript是基础。本书的后续章节都需要读者能熟练使用JavaScript。JavaScript是一门面向对象的强大的前端脚本语言，如果要深入学习JavaScript，可能需要一本书的厚度来介绍，这并不是本书的重点。因此，如果你对JavaScript没有任何基础，建议学习完本书的准备章节后，先系统地学习一下JavaScript语言基础，再继续学习本书后续的Vue章节。

本节将只介绍JavaScript核心、基础的一些概念。

1.4.1　为什么需要 JavaScript

如果将一个网页类比为一个人，HTML构建了其骨架，CSS为其着装打扮，而JavaScript则为其

赋予了灵魂。不夸张地说，JavaScript就是网页应用的灵魂。通过前面的学习，我们知道，HTML和CSS的主要作用是对网页的渲染进行布局和调整，要使得网页拥有强大的功能并且可以与用户进行复杂的交互，都需要使用JavaScript来完成。

首先，JavaScript能够动态改变HTML组件的内容。创建一个名为js.html的文件，在其中编写如下示例代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        var count = 0
        function clickFunc() {
            document.getElementById("h1").innerText = `${++count}`
        }
    </script>
</head>
<body>
    <div style="text-align: center;">
        <h1 id="h1" style="font-size: 40px;">数值:0</h1>
        <button style="font-size: 30px; background-color: burlywood;"
onclick="clickFunc()">点击</button>
    </div>
</body>
</html>
```

上面的代码中使用到了几个核心的知识点，在HTML标签中可以直接内嵌CSS样式表，为其设置style属性即可，内嵌的样式表要比外联的样式表优先级更高。button标签是HTML中定义按钮的标签，其中onclick属性可以设置一段JavaScript代码，当用户单击按钮组件时会调用这段代码。如以上代码所示，当用户单击按钮时，我们让其执行了clickFunc函数。clickFunc函数定义在script标签中，其实现了简单的计数功能，document对象是当前的文档对象，调用其getElementById方法可以通过元素标签的id属性的值来获取对应元素，调用innerText可以对元素标签内的文本进行设置。运行代码，可以看到网页上渲染了一个标题和按钮，通过单击按钮，标题上显示的数字会进行累加，如图1-14所示。

图 1-14　使用 JavaScript 实现计数器

使用JavaScript也可以方便地对标签元素的属性进行设置和修改。例如，我们在页面中再添加一个图片元素，通过单击按钮来设置其显示和隐藏状态：

HTML标签：

```
<div id="img" style="visibility: visible;">
    <img src="demo.png" width="200px">
</div>
```

JavaScript代码：

```
<script>
    var count = 0
    function clickFunc() {
        document.getElementById("h1").innerText = `${++count}`
        document.getElementById("img").style.visibility = count % 2 == 0 ?
"visible" : "hidden"
    }
</script>
```

可以看到，使用JavaScript获取标签的属性非常简单，直接使用点语法即可。同理，我们也可以通过这种方式来灵活地控制网页上元素的样式，只需要修改元素的style属性即可。运行上面的代码，在网页中单击按钮，可以看到图片元素会交替地进行显示与隐藏。

使用JavaScript也可以非常容易地对HTML文档中的元素进行增删，有时一个非常简单的HTML文档能够实现非常复杂的页面，其实都是通过JavaScript来动态渲染的。

1.4.2 JavaScript 语法简介

JavaScript语言的语法非常简单，入门很容易，对于开发者来说上手也非常快。其语法不像某些强类型语言那样严格，语句格式和变量类型都非常灵活。

1. 变量的定义

JavaScript使用var或let来进行变量的定义。其使用var定义和let定义会使得变量的作用域不同。在定义变量时，无须关心变量的类型，例如：

```
<script>
    var a = 100              // 定义变量a，其存储的值为数值100
    var b = "HelloWorld"     // 定义变量b，其存储的值为字符串
    var c = {
        name:"abc"
    }                        // 定义键值对变量 c
    var d = [1, 2, 3]        // 定义列表变量d
    var e = false            // 定义布尔变量e
</script>
```

JavaScript中的注释规则与传统的C语言类似，我们一般使用“//”来定义注释。

2. 表达式

几乎在任何编程语言中都存在表达式，表达式由运算符与运算数构成。运算数可以是任意类型的数据，也可以是任意的变量，只要其能支持我们指定的运算。JavaScript支持很多常规的运算符，例如算术运算符+、–、*、/等，比较运算符<、>、<=、<=等，示例如下：

```
var m = 1 + 1                // 算术运算
var n = 10 > 5               // 比较运算
var o = false && false       // 逻辑运算
var p = 1 << 1               // 位运算
```

3. 函数的定义与调用

函数是程序的功能单元，在JavaScript中定义函数有两种方式，一种是使用function关键字进行定义，另一种是使用箭头函数的方式进行定义。无论哪种方式定义的函数，其调用方式都是一样的，示例如下：

```
function func1(param) {
    console.log("执行了func1函数" + param);
}
var func2 = (param) => {
    console.log("执行了func2函数" + param);
}
func1("hello")
func2("world")
```

运行上面的代码，在VSCode开发工具的控制台可以看到输出的信息。console.log函数用来向控制台输出信息。

4. 条件分支语句

条件语句是JavaScript进行逻辑控制的重要语句，当程序需要根据条件是否成立来分别执行不同的逻辑时，就需要使用条件语句，JavaScript中的条件语句使用if和else关键词来实现，示例如下：

```
var i = 0
var j = 1
if (i > j) {
    console.log("i > j")
} else if (i == j) {
    console.log("i == j")
} else {
    console.log("i < j")
}
```

JavaScript中也支持使用switch和case关键字多分支语句，示例如下：

```
var u = 0
switch (u) {
    case 0:
        console.log("0")
        break
    case 1:
        console.log("1")
        break
    default:
        console.log("-")
}
```

5. 循环语句

循环语句用来重复执行某段代码逻辑，JavaScript中支持while型循环和for型循环，示例代码如下：

```
var v = 10
while (v > 0) {
    v -= 1
    console.log(v)
}
for(v = 0 ; v < 10; v ++) {
    console.log(v)
}
```

除此之外，JavaScript还有许多非常强大的语法与面向对象能力，我们在后面的章节中会更加详细地介绍。

1.5 渐进式开发框架 Vue

Vue的定义为渐进式的JavaScript框架。所谓渐进式，是指其被设计为可以自底向上逐层应用。我们可以只使用Vue框架中提供的某层的功能，也可以与其他第三方库整合使用。当然，Vue本身也提供了完整的工具链，使用其全套功能进行项目的构建也非常简单。

在使用Vue之前，需要掌握基础的HTML、CSS和JavaScript技能，如果你对本章前面介绍的内容都已经掌握，那么对于后面Vue使用的相关例子会非常容易理解。Vue的渐进式性质使其使用方式变得非常灵活，在使用时，我们可以使用其完整的框架，也可以只使用部分功能。

1.5.1 第一个 Vue 应用

在学习和测试Vue的功能时，我们可以直接使用CDN的方式来进入Vue框架。本书将全部采用Vue 3.0.x的版本来编写示例。首先，使用VSCode开发工具创建一个名为Vue1.html的文件，在其中编写如下模板代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue3 Demo</title>
    <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
</body>
</html>
```

其中，我们在head标签中加入了一个script标签，采用CDN的方式引入了Vue3的新版本。以之

前编写的计数器应用为例，我们尝试使用Vue的方式来实现它。首先在body标签中添加一个标题和按钮，代码如下：

```
<div style="text-align: center;" id="Application">
    <h1>{{ count }}</h1>
    <button v-on:click="clickButton">点击</button>
</div>
```

上面使用到了一些特殊的语法，例如在h1标签内部使用了Vue的变量替换功能，{{ count }}是一种特殊语法，其会将当前Vue组件中定义的count变量的值替换过来，v-on:click属性用来进行组件的单击事件绑定，上面的代码将单击事件绑定到了clickButton函数上，这个函数也是定义在Vue组件中的。定义Vue组件非常简单，我们可以在body标签下添加一个script标签，在其中编写如下代码：

```
<script>
    // 定义一个Vue组件，名为App
    const App = {
        // 定义组件中的数据
        data() {
            return {
                // 目前我们只用到count数据
                count:0
            }
        },
        // 定义组件中的函数
        methods: {
            // 实现单击按钮的方法
            clickButton() {
                this.count = this.count + 1
            }
        }
    }
    // 将Vue组件绑定到页面上id为Application的元素上
    Vue.createApp(App).mount("#Application")
</script>
```

如以上代码所示，我们定义Vue组件时实际上定义了一个JavaScript的对象，其中data方法用来返回组件所需要的数据，methods属性用来定义组件所需要的方法函数。在浏览器中运行上面的代码，当单击页面中的按钮时，计数器会自动增加。可以看到，使用Vue实现的计数器应用比使用JavaScript直接操作HTML元素方便得多，我们不需要获取指定的组件，也不需要修改组件中的文本内容，通过Vue这种绑定的编程方式，只需要专注于数据逻辑，当数据本身修改时，绑定这些数据的元素也会同步修改。

1.5.2　范例：实现一个简单的用户登录页面

本节尝试使用Vue来构建一个简单的登录页面。在练习之前，我们先来分析一下需要完成哪些工作：

（1）登录页面需要有标题，用来提示用户当前的登录状态。

（2）在未登录时，需要有两个输入框以及登录按钮供用户输入账号、密码和进行登录操作。

（3）在登录完成后，输入框需要隐藏，需要提供按钮让用户登出。

要完成上面列出的3个功能点，如果使用原生的JavaScript DOM操作会有些复杂，借助Vue的单双向绑定和条件渲染功能，完成这些需求则会非常容易。

首先创建一个名为loginDemo.html的文件，为其添加HTML通用的模板代码，并通过CND的方式引入Vue。之后，在其body标签中添加如下代码：

```
    <div id="Application" style="text-align: center;">
        <h1>{{title}}</h1>
        <div v-if="noLogin">账号：<input v-model="userName" type="text" /></div>
        <div v-if="noLogin">密码：<input v-model="password" type="password"
/></div>
        <div v-on:click="click" style="border-radius: 30px;width: 100px; margin:
20px auto; color: white; background-color: blue;">{{buttonTitle}}</div>
    </div>
```

上面的代码中，v-if是Vue提供的条件渲染功能，其指定的变量如果为true，则渲染这个元素，否则不渲染。v-model用来进行双向绑定，当输入框中的文字变化时，其会将变化同步到绑定的变量上，同样，当我们对变量的值进行改变时，输入框中的文本也会对应变化。

实现JavaScript代码如下：

```
    <script>
        const App = {
            data () {
                return {
                    title:"欢迎您：未登录",
                    noLogin:true,
                    userName:"",
                    password:"",
                    buttonTitle:"登录"
                }
            },
            methods: {
                click() {
                    if (this.noLogin) {
                        this.login()
                    } else {
                        this.logout()
                    }
                },
                // 登录
                login() {
                    // 判断账号、密码是否为空
                    if (this.userName.length > 0 && this.password.length > 0) {
                        // 登录提示后刷新页面
                        alert(`userNmae:${this.userName}
password:${this.password}`)
```

```
                this.noLogin = false
                this.title = `欢迎您:${this.userName}`
                this.buttonTitle = "注销"
                this.userName = ""
                this.password = ""
            } else {
                alert("请输入账号密码")
            }
        },
        // 登出
        logout() {
            // 清空登录数据
            this.noLogin = true
            this.title = `欢迎您:未登录`
            this.buttonTitle = "登录"
        }
    }
}
Vue.createApp(App).mount("#Application")
</script>
```

运行上面的代码，未登录时效果如图1-15所示。当输入了账号和密码完成登录后，效果如图1-16所示。

图 1-15　简易登录页面 1

图 1-16　简易登录页面 2

1.5.3　Vue 3 的新特性

如果你之前接触过前端开发，相信Vue框架对于你来说并不陌生。Vue 3的发布无疑是Vue框架的一次重大改进。一款优秀的前端开发框架的设计一定要遵循一定的设计原理，Vue 3的设计目标是：

（1）更小的尺寸和更快的速度。
（2）更加现代化的语法特性，加强TypeScript的支持。
（3）在API设计方面，增强统一性和一致性。

（4）提高前端工程的可维护性。

（5）支持更多、更强大的功能，提高开发者的效率。

上面列举了Vue 3的核心设计目标，相较于Vue 2版本，Vue 3有哪些重大的更新点呢？本节我们就来简单介绍一下。

首先，在Vue 2时代，最小化被压缩的Vue核心代码约为20KB，目前Vue 3的压缩版只有10KB，大小足足减少了一半。在前端开发中，依赖模块越小，意味着更少的流量和更快的速度，在这方面，Vue 3的确表现优异。

在Vue 3中，对虚拟DOM的设计也进行了优化，使得引擎可以更加快速地处理局部的页面元素修改，在一定程度上提升了代码的运行效率。同时，Vue 3也配套进行了更多编译时的优化，例如将插槽编译为函数等。

在代码语法层面上，相较于Vue 2，Vue 3有比较大的变化。其基本弃用了“类”风格的API，而推广采用“函数”风格的API，以便更好地对TypeScript进行支持。这种编程风格更有利于组件的逻辑复用，例如Vue 3组件中心引入的setup（组合式API）方法，可以让组件的逻辑更加聚合。

Vue 3中也添加了一些新的组件，比如Teleport组件（有助于开发者将逻辑关联的组件封装在一起），这些新增的组件提供了更加强大的功能，以便于开发者对逻辑的复用。

总之，在性能方面，Vue 3无疑完胜Vue 2，同时打包后的体积也更小。在开发者编程方面，Vue 3基本是向下兼容的，开发者无须过多的额外学习成本，并且Vue 3对功能方面的扩展对于开发者来说也更加友好。

关于Vue 3更详细的内容与新特性的使用方法，后面的章节会逐步向读者介绍。

1.5.4 为什么要使用 Vue 框架

在真正开始学习Vue之前，还有一个问题至关重要，那就是我们为什么要学习它。

首先，做前端开发，一定要使用一款框架，这就像生产产品的工厂有一套完整的流水线一样。在学习阶段，我们可以直接使用HTML、CSS和JavaScript开发出一些简单的静态页面，但是要做大型的商业应用，要完成的代码量会非常大，要编写的功能函数会非常多，而且对于交互复杂的项目来说，如果真的不使用任何框架来开发的话，后期维护和扩展将会非常困难。

既然一定要使用框架，那么我们为什么要选择Vue呢？在互联网Web时代早期，前后端的界限还比较模糊，有一个名为jQuery的JavaScript框架非常流行，其内部封装了大量的JavaScript函数，可以帮助开发者操作DOM并且提供了事件处理、动画和网络相关接口。当时的前端页面更多是用来展示，因此使用jQuery框架足够应付所需要进行的逻辑交互操作。后来随着互联网的飞速发展，前端网站的页面越来越复杂，2009年诞生了一款名为AngularJS的前端框架，此框架的核心是响应式与模块化，其使得前端页面的开发方式发生了变革，前端可以自行处理非常复杂的业务逻辑，前后端的职责开始逐渐分离，前端从页面展示向单页面应用发展。

AngularJS虽然强大，但其缺点也十分明显，总结如下：

（1）学习曲线陡峭，入门难度高。

（2）灵活性很差，这意味着如果要使用AngularJS，就必须按照其规定的一套构造方式来开发应用，要完整地使用其一整套的功能。

（3）由于框架本身很庞大，使得速度和性能会略差。

（4）在代码层面上，某些API设计复杂，使用麻烦。

只要AngularJS有上述问题，就一定会有新的前端框架来解决这些问题，Vue和React这两个框架就此诞生了。

Vue和React在当下前端项目开发中平分秋色，它们都是非常优秀的现代化前端框架。从设计上，它们有很多相似之处，比如相较于功能齐全的AngularJS而言，它们都是“骨架”类的框架，即只包含基础的核心功能，路由、状态管理等功能都是靠分离的插件来支持的。并且在逻辑上，Vue和React都是基于虚拟DOM树的，改变页面真实的DOM要比虚拟DOM的更改性能开销大很多，因此Vue和React的性能都非常优秀。Vue和React都引导采用组件化的方式进行编程，模块间通过接口进行连接，方便维护与扩展。

当然，Vue与React也有很多不同之处，Vue的模板编写采用的是类似HTML的模板方式，写起来与标准的HTML非常像，只是多了一些数据绑定或事件交互的方法，入手非常简单。而React则是采用JSX的方式编写模板，虽然这种编写方式提供的功能更加强大一些，但是JavaScript混合XML的语言使得代码看上去非常复杂，阅读起来也比较困难。Vue与React还有一个很大的区别在于组件状态管理，Vue的状态管理本身非常简单，局部的状态只要在data中进行定义，其默认就被赋予了响应性，在需要修改时直接将对应属性更改即可，对于全局的状态也有Vuex模块进行支持。在React中，状态不能直接修改，需要使用setState方法进行更改，从这一点上看，Vue的状态管理更加简洁一些。

总之，如果你想尽快掌握前端开发的核心技能并上手开发大型商业项目，Vue一定不会让你失望。

1.6　小结与练习

本章是我们进入Vue学习的准备章节，在学习Vue框架之前，首先需要熟练应用前端3剑客（HTML、CSS和JavaScript）。同时，我们对Vue的使用也有了初步的体验，相信你已经体会到了Vue在开发中为我们带来的便利与高效。

通过本章的学习，请你尝试回答下面的问题，如果每道问题在你的心中都有了清晰的答案，那么恭喜你过关成功，快快开始下一章的学习吧！

练习1：在网页开发中，HTML、CSS和JavaScript分别起到了什么样的作用？

温馨提示：可以从布局、样式和逻辑处理方面思考。

练习2：如何动态地改变网页元素的样式或内容，请你尝试在不使用Vue的情况下，手动实现本章1.5.2节的登录页面。

温馨提示：尝试使用JavaScript的DOM操作来重写示例工程。

练习3：数据绑定在Vue中如何使用？什么是单向绑定？什么是双向绑定？

温馨提示：结合本章1.5.2节的示例进行分析。

练习4：通过对Vue示例工程的体验，你认为使用Vue开发前端页面的优势有哪些？

温馨提示：可以从数据绑定、方法绑定条件和循环渲染以及Vue框架的渐进式性质进行思考。

第2章

Vue 模板应用

模板是Vue框架中的重要组成部分，Vue采用了基于HTML的模板语法，因此对于大多数开发者来说上手会非常容易。在Vue的分层设计思想中，模板属于视图层，有了模板功能，开发者可以方便地将项目组件化，也可以方便地封装定制化的通用组件。在编写组件时，模板的作用是让开发者将重心放在页面布局渲染上，而不需要关心数据逻辑。同样，在Vue组件内部编写数据逻辑代码时，也无须关心视图的渲染。

本章将着重学习Vue框架的模板部分，现在就奔向我们Vue学习之路上的第一个目标吧：游刃有余地使用模板。

通过本章，你将学习到：

- 基础的模板使用语法。
- 模板中参数的使用。
- Vue 指令相关用法。
- 使用缩写指令。
- 灵活使用条件语句与循环语句。

2.1　模板基础

在第1章我们已经体验过模板，对于普通的HTML文档，若要在数据变化时对其进行页面的更新，则需要通过JavaScript的DOM操作来获取指定的元素，再对其属性或内部文本进行修改，操作起来十分烦琐且容易出错。如果使用了Vue的模板语法，则事情会变得非常简单，我们只需要将要变化的值定义成变量，之后将变量插入HTML文档指定的位置即可。当数据发生变化时，使用此变量的所有组件都会同步更新，这就使用到了Vue模板中的插值技术。学习模板，我们先从学习插值开始。

2.1.1　模板插值

首先，我们创建一个名为tempText.html的文件，在其中编写HTML文档的常规代码。之后在body标签中添加一个元素供我们测试使用，代码如下：

```
<div style="text-align: center;">
    <h1>这里是模板的内容:1次点击</h1>
    <button>按钮</button>
</div>
```

如果在浏览器中运行上面的HTML代码，你会看到网页中渲染出了一个标题和一个按钮，但是单击按钮并没有任何效果（截至目前，我们并没有写什么逻辑代码）。现在，让我们为这个网页增加一些动态功能，很简单：单击按钮，改变数值。引入Vue框架，并通过Vue组件来实现这个计数器功能，完整的示例代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>模板插值</title>
    <script src="https://unpkg.com/vue@next"></script>
</head>
```

```
<body>
    <div id="Application" style="text-align: center;">
        <h1>这里是模板的内容:{{count}}次点击</h1>
        <button v-on:click="clickButton">按钮</button>
    </div>
    <script>
        // 定义一个Vue组件，名为App
        const App = {
            // 定义组件中的数据
            data() {
                return {
                    // 目前我们只用到count数据
                    count:0
                }
            },
            // 定义组件中的函数
            methods: {
                // 实现单击按钮的方法
                clickButton() {
                    this.count = this.count + 1
                }
            }
        }
        // 将Vue组件绑定到页面上id为Application的元素上
        Vue.createApp(App).mount("#Application")
    </script>
</body>
</html>
```

在浏览器中运行上面的代码，单击页面中的按钮。可以看到页面中标题的文本也在不断变化。如以上代码所示，在 HTML 的标签中使用“{{}}”可以进行变量插值，这是 Vue 中基础的模板语法，其可以将当前组件中定义的变量的值插入指定位置，并且这种插值会默认实现绑定的效果，即当我们修改了变量的值时，其可以同步地反馈到页面的渲染上。

某些情况下，某些组件的渲染是由变量控制的，但是我们想让它一旦渲染后就不能够再被修改，这时可以使用模板中的v-once指令实现，被这个指令设置的组件在进行变量插值时只会插值一次，例如：

```
<h1 v-once>这里是模板的内容:{{count}}次点击</h1>
```

在浏览器中再次实验，可以发现网页中指定的插值位置被替换成了文本“0”后，无论我们再怎么单击按钮，标题也不会再改变。

还有一点需要注意，如果要插值的文本为一段HTML代码，则直接使用双括号就不太好使了，双括号会将其内的变量解析成纯文本。例如，定义Vue组件App中的数据如下：

```
data() {
    return {
        count:0,
        countHTML:"<span style='color:red;'>0</span>"
    }
```

```
}
```

如果使用双括号插值的方式将HTML代码插入，最终的效果会将其以文本的方式渲染出来，代码如下：

```
<h1 v-once>这里是模板的内容:{{countHTML}}次点击</h1>
```

运行效果如图2-1所示。

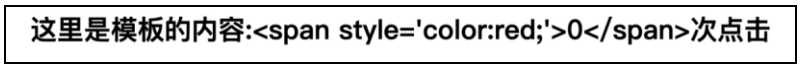


图 2-1　使用双括号进行 HTML 插值

这种效果明显不符合我们的预期，对于HTML代码插值，我们需要使用v-html指令来完成，示例代码如下：

```
<h1 v-once>这里是模板的内容:<span v-html="countHTML"></span>次点击</h1>
```

V-html指令可以指定一个Vue变量数据，其会通过HTML解析的方式将原始HTML替换到其指定的标签位置，如以上代码运行后的效果如图2-2所示。

图 2-2　使用 v-html 进行 HTML 插值

前面介绍了如何在标签内部进行内容的插值，我们知道，标签除了其内部的内容外，本身的属性设置也是非常重要的，例如我们可能需要动态改变标签的style属性，从而实现元素渲染样式的修改。在Vue中，我们可以使用属性插值的方式做到标签属性与变量的绑定。

对于标签属性的插值，Vue中不再使用双括号的方式，而是使用v-bind指令，示例代码如下：

```
<h1 v-bind:id="id1">这里是模板的内容:{{count}}次点击</h1>
```

定义一个简单的CSS样式如下：

```
#h1 {
    color: red;
}
```

再添加一个名为id1的Vue组件属性，代码如下：

```
data() {
    return {
        count:0,
        countHTML:"<span style='color:red;'>0</span>",
        id1:"h1"
    }
}
```

运行代码，可以看到我们已经将 id 属性动态地绑定到了指定的标签中，当 Vue 组件中 id1 属性的值发生变化时，其也会动态地反映到 h1 标签上，我们通过这种动态绑定的方式灵活地更改标签的样式表。v-bind 指令同样适用于其他 HTML 属性，只需要在其中使用冒号加属性名的方式指

定即可。

其实，无论是双括号方式的标签内容插值还是v-bind方式的标签属性插值，除了可以直接将变量插值外，也可以使用基本的JavaScript表达式，例如：

```
<h1 v-bind:id="id1">这里是模板的内容:{{count + 10}}次点击</h1>
```

上面的代码运行后，页面上渲染的数值是count属性增加10之后的结果。有一点需要注意，所有插值的地方如果使用表达式，则只能使用单个表达式，否则会产生异常。

2.1.2 模板指令

本质上，Vue中的模板指令也是HTML标签属性，其通常由前缀“v-”开头，例如前面使用的v-bind、v-once等都是指令。

大部分指令都可以直接设置为JavaScript变量或单个的JavaScript表达式。首先创建一个名为directives.html的测试文件，在其中编写HTML的通用代码后引入Vue框架，之后在body标签中添加如下代码：

```
<div id="Application">
    <h1 v-if="show">标题</h1>
</div>
<script>
    const App = {
        data() {
            return {
                show:false
            }
        }
    }
    Vue.createApp(App).mount("#Application")
</script>
```

如以上代码所示，其中v-if是一个简单的选择渲染指令，设置为布尔值true时，当前标签元素才会被渲染。

某些特殊的Vue指令也可以指定参数，例如v-bind和v-on指令，对于可以添加参数的指令，参数和指令使用冒号进行分隔，例如：

```
v-bind:style
V-on:click
```

指令的参数本身也可以是动态的，例如我们可以定义区分id选择器和类选择器来定义不同的组件样式，之后动态地切换组件的属性，示例如下：

CSS样式：

```
#h1 {
    color:red;
}
.h1 {
    color:blue
```

```
}
```

HTML标签定义如下：

```
<h1 v-bind:[prop]="name" v-if="show">标题</h1>
```

在Vue组件中定义属性数据如下：

```
const App = {
    data() {
        return {
            show:true,
            prop:"class",
            name:"h1"
        }
    }
}
```

在浏览器中运行上面的代码，可以看到h1标签被正确地绑定了class属性。

在参数后面还可以为Vue中的指令增加修饰符，修饰符会为Vue指令增加额外的功能，以一个常见的应用场景为例，在网页中，如果有可以输入信息的输入框，通常不希望用户在首尾输入空格符，通过Vue的指令修饰符，可以很容易地实现自动去除首尾空格符的功能，示例代码如下：

```
<input v-model.trim="content">
```

如以上代码所示，我们使用v-model指令将输入框的文本与content属性进行绑定，当用户在输入框中输入的文本首尾有空格符时，以及输入框失去焦点时，Vue会自动帮我们去掉这些首尾空格。

你应该已经体会到了Vue指令的灵活与强大之处，最后我们将介绍Vue中常用的两个缩写。在Vue应用开发中，v-bind和v-on两个指令的使用非常频繁，对于这两个指令，Vue也为开发者提供了更加高效的缩写方式，对于v-bind指令，我们可以直接将其v-bind前缀省略，直接使用冒号加属性名的方式进行绑定，例如v-bind:id ="id"可以缩写为如下模样：

```
:id="id"
```

对于v-on类的事件绑定指令，可以将前缀v-on:使用@符替代，例如v-on:click="myFunc"指令可以缩写成如下模样：

```
@click="myFunc"
```

在后面的学习中你会体验到，有了这两个缩写功能，将大大提高Vue应用的编写效率。

2.2　条 件 渲 染

条件渲染是Vue控制HTML页面渲染的方式之一。很多时候，我们都需要通过条件渲染的方式来控制HTML元素的显示和隐藏。在Vue中，要实现条件渲染，可以使用v-if相关的指令，也可以使用v-show相关的指令。本节将细致地探讨这两种指令的使用。

2.2.1 使用 v-if 指令进行条件渲染

v-if指令在之前的测试代码中也简单地使用过，简单来讲，其可以有条件地选择是否渲染一个HTML元素。v-if指令可以设置为一个JavaScript变量或表达式，当变量或表达式为真值时，其指定的元素才会被渲染。为了方便代码测试，我们可以新建一个名为condition.html的测试文件，在其中编写代码。

简单的条件渲染示例如下：

```
<h1 v-if="show">标题</h1>
```

在上面的代码中，只有当show变量的值为真时当前标题元素才会被渲染，Vue模板中的条件渲染指令v-if类似于JavaScript编程语言中的if语句。我们都知道在JavaScript中，if关键字可以和else关键字结合使用组成if-else块，在Vue模板中也可以使用类似的条件渲染逻辑，v-if指令可以和v-else指令结合使用，示例如下：

```
<h1 v-if="show">标题</h1>
<p v-else>如果不显示标题就显示段落</p>
```

运行代码可以看到，标题元素与段落元素是互斥出现的，如果根据条件渲染出了标题元素，则不会再渲染出段落元素，如果没有渲染出标题元素，则会渲染出段落元素。需要注意，在将v-if与v-else结合使用时，设置了v-else指令的元素必须紧跟在v-if或v-else-if指令指定的元素后面，否则其不会被识别到。例如下面的代码，运行后效果如图2-3所示。

图 2-3 条件渲染示例

```
<h1 v-if="show">标题</h1>
<h1>Hello</h1>
<p v-else>如果不显示标题就显示段落
</p>
```

其实，如果在 VSCode 中编写了上面的代码并运行，VSCode 开发工具的控制台也会打印出相关异常信息提示你 v-else 指令使用错误，如图 2-4 所示。

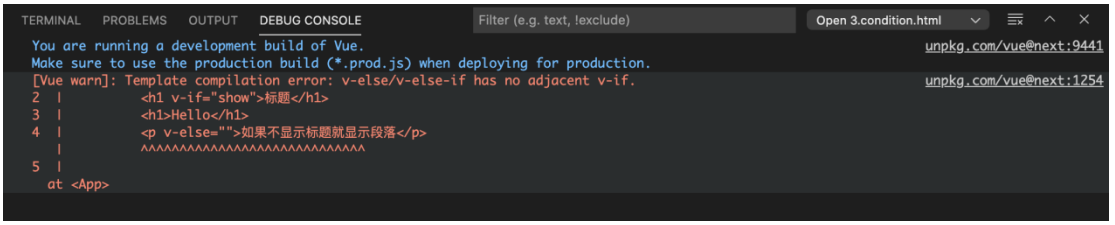


图 2-4 VSCode 控制台打印出的异常提示

在v-if与v-else之间，我们还可以插入任意个v-else-if来实现多分支渲染逻辑。在实际应用中，

多分支渲染逻辑也很常用，例如根据学生的分数来将成绩进行分档，就可以使用多分支逻辑，示例代码如下：

```
<h1 v-if="mark == 100">满分</h1>
<h1 v-else-if="mark > 60">及格</h1>
<h1 v-else>不及格</h1>
```

v-if指令的使用必须添加到一个HTML元素上，如果我们需要使用条件同时控制多个标签元素的渲染，有两种方式可以实现。

（1）使用div标签对要进行控制的元素进行包装，示例如下：

```
<div v-if="show">
    <p>内容</p>
    <p>内容</p>
    <p>内容</p>
</div>
```

（2）使用template标签对元素进行分组，示例如下：

```
<template v-if="show">
    <p>内容</p>
    <p>内容</p>
    <p>内容</p>
</template>
```

通常，更推荐使用template分组的方式来控制一组元素的条件渲染逻辑，因为在HTML渲染元素时，使用div包装组件后，div元素本身会被渲染出来，而使用template进行分组的组件渲染后并不会渲染template标签本身。我们可以通过Chrome浏览器来验证这种特性，在Chrome浏览器中按F12按键可以打开开发者工具窗口，也可以通过单击菜单栏中的“更多工具”→“开发者工具”来打开此窗口，如图2-5所示。

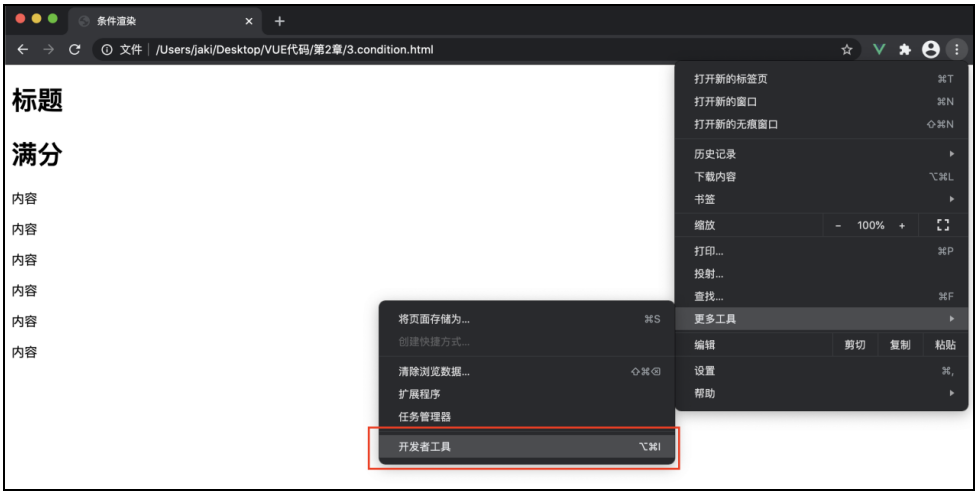


图 2-5　打开 Chrome 的开发者工具

在开发中工具窗口的Elements栏目中可以看到使用div和使用template标签对元素组合包装进

行条件渲染的异同，如图2-6所示。

图 2-6　使用 Chrome 开发者工具分析渲染情况

2.2.2　使用 v-show 指令进行条件渲染

v-show指令的基本用法与v-if类似，其也是通过设置条件的值的真假来决定元素的渲染情况的。示例如下：

```
<h1 v-show="show">v-show标题在这里</h1>
```

与v-if不同的是，v-show并不支持template模板，同样也不可以和v-else结合使用。

虽然v-if与v-show的用法非常相似，但是它们的渲染逻辑是天差地别的。

从元素本身的存在性来说，v-if才是真正意义上的条件渲染，其在条件变换的过程中，组件内部的事件监听器都会正常地执行，子组件也会正常地被销毁或重建。同时，v-if采取的是懒加载的方式进行渲染，如果初始条件为假，则关于这个组件的任何渲染工作都不会进行，直到其绑定的条件为真时，才会真正开始渲染此元素。

v-show指令的渲染逻辑只是一种视觉上的条件渲染 实际上无论v-show指令设置的条件是真是假，当前元素都会被渲染，v-show指令只是简单地通过切换元素CSS样式中的display属性来实现展示效果。

我们可以通过Chrome浏览器的开发者工具来观察v-if与v-show指令的渲染逻辑，示例代码如下：

```
<h1 v-if="show">v-if标题在这里</h1>
<h1 v-show="show">v-show标题在这里</h1>
```

当条件为假时，可以看到，v-if指定的元素不会出现在HTML文档DOM结构中，而v-show指定的元素依然会存在，如图2-7所示。

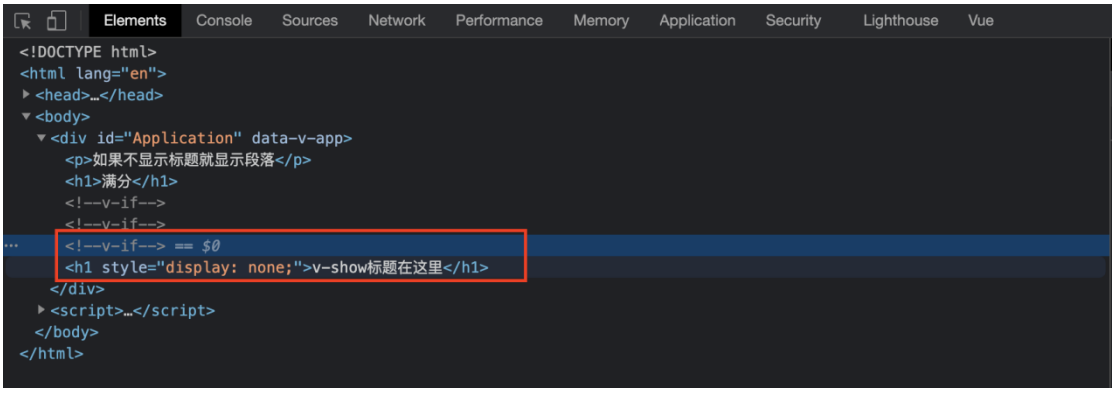


图 2-7　v-if 与 v-show 的区别

由于v-if与v-show这两种指令的渲染原理不同，通常v-if指令有更高的切换性能消耗，而v-show指令有更高的初始渲染性能消耗。在实际开发中，如果组件的渲染条件会比较频繁地切换，则建议使用v-show指令来控制，如果组件的渲染条件在初始指定后就很少变化，则建议使用v-if指令控制。

2.3　循 环 渲 染

在网页中，列表是非常常见的一种组件。在列表中，每一行元素都有相似的UI，只是其填充的数据有所不同，使用Vue中的循环渲染指令，我们可以轻松地构建出列表视图。

2.3.1　v-for 指令的使用方法

在Vue中，v-for指令可以将一个数组中的数据渲染为列表视图。v-for指令需要设置为一种特殊的语法，其格式如下：

```
item in list
```

在上面的格式中，in为语法关键字，其也可以替换为of。

在v-for指令中，item是一个临时变量，其为列表中被迭代出的元素名，list是列表变量本身。我们可以新建一个名为for.html的测试文件，在其body标签中编写如下核心代码：

```
<body>
    <div id="Application">
        <div v-for="item in list">
            {{item}}
        </div>
    </div>
    <script>
        const App = {
            data() {
                return {
                    list:[1,2,3,4,5]
```

```
                }
            }
        }
        Vue.createApp(App).mount("#Application")
    </script>
</body>
```

运行代码，可以看到网页中正常地渲染出了5个div组件，如图2-8所示。

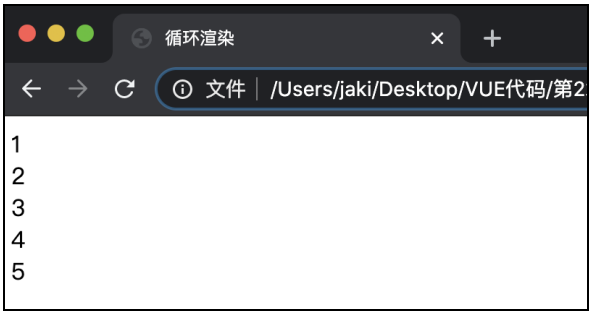


图 2-8　循环渲染效果图

更多时候，我们需要渲染的数据都是对象数据，使用对象来对列表元素进行填充，例如定义联系人对象列表如下：

```
list:[
    {
        name: "珲少",
        num: "151xxxxxxxx"
    },
    {
        name: "Jaki",
        num: "151xxxxxxxx"
    },
    {
        name: "Lucy",
        num: "151xxxxxxxx"
    },
    {
        name: "Monki",
        num: "151xxxxxxxx"
    },
    {
        name: "Bei",
        num: "151xxxxxxxx"
    }
]
```

修改要渲染的HTML标签结构如下：

```
<div id="Application">
    <ul>
        <li v-for="item in list">
```

```
            <div>{{item.name}}</div>
            <div>{{item.num}}</div>
        </li>
    </ul>
</div>
```

运行代码，效果如图2-9所示。

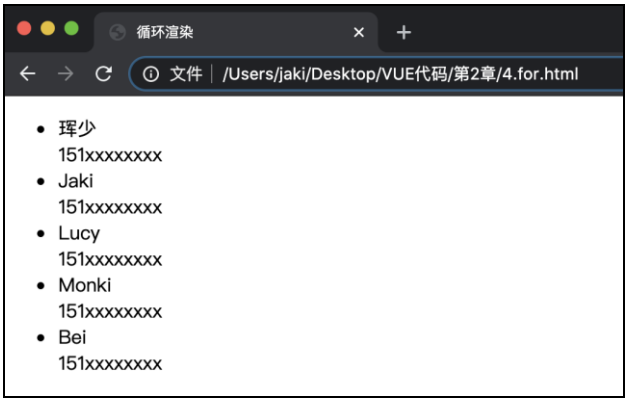


图 2-9　使用对象数据进行循环渲染

在v-for指令中，我们也可以获取到当前遍历项的索引，示例如下：

```
<ul>
    <li v-for="(item,index) in list">
        <div>{{index + "." + item.name}}</div>
        <div>{{item.num}}</div>
    </li>
</ul>
```

需要注意，index索引的取值是从0开始的。

在上面的示例代码中，v-for指令遍历的为列表，实际上我们也可以对一个JavaScript对象进行v-for遍历。在JavaScript中，列表本身也是一种特殊的对象，我们使用v-for对对象进行遍历时，指令中的第1个参数为遍历的对象中的属性的值，第2个参数为遍历的对象中的属性的名字，第3个参数为遍历的索引。首先，定义对象如下：

```
person: {
    name: "珲少",
    age: "00",
    num: "151xxxxxxxx",
    emali: "xxxx@xx.com"
}
```

我们使用有序列表来承载person对象的数据，代码如下：

```
<ol>
    <li v-for="(value,key,index) in preson">
        {{key}}:{{value}}
    </li>
</ol>
```

运行代码，效果如图2-10所示。

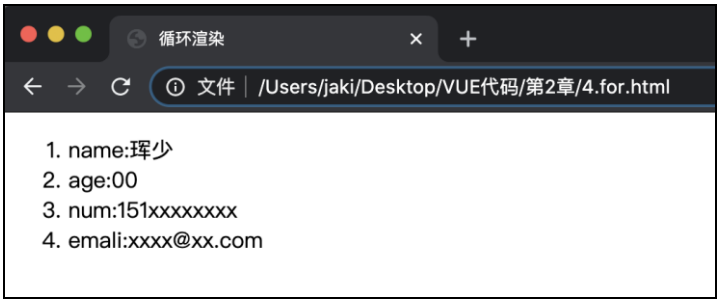


图 2-10　将对象数据渲染到页面

需要注意，在使用v-for指令进行循环渲染时，为了更好地对列表项进行重用，我们可以将其key属性绑定为一个唯一值的，代码如下：

```
<ol>
    <li v-for="(value,key,index) in person" :key="index">
        {{key}}:{{value}}
    </li>
</ol>
```

2.3.2　v-for 指令的高级用法

当我们使用v-for对列表进行循环渲染后，实际上就实现了对这个数据对象的绑定。当我们调用下面这些函数对列表数据对象进行更新时，视图也会对应地更新：

```
push()          // 向列表尾部追加一个元素
pop()           // 删除列表尾部的一个元素
shift()         // 向列表头部插入一个元素
unshift()       // 删除列表头部的一个元素
splice()        // 对列表进行分割操作
sort()          // 对列表进行排序操作
reverse()       // 对列表进行逆序
```

首先在页面上添加一个按钮来演示列表的逆序操作：

```
<button @click="click">
    逆序
</button>
```

定义Vue函数如下：

```
methods: {
    click() {
        this.list.reverse()
    }
}
```

运行代码，可以看到当单击页面上的按钮时，列表元素的渲染顺序会进行正逆切换。当我们

需要对整个列表都进行替换时，直接对列表变量重新赋值即可。

在实际开发中，原始的列表数据往往并不适合直接渲染到页面，v-for指令支持在渲染前对数据进行额外的处理，修改标签如下：

```
<ul>
    <li v-for="(item,index) in handle(list)">
        <div>{{index + "." + item.name}}</div>
        <div>{{item.num}}</div>
    </li>
</ul>
```

上面的代码中，handle为定义的处理函数，在进行渲染前，通过这个函数来对列表数据进行处理。例如，我们可以使用过滤器来进行列表数据的过滤渲染，实现handle函数如下：

```
handle(l) {
    return l.filter(obj => obj.name != "珲少")
}
```

当需要同时循环渲染多个元素时，与v-if指令类似，常用的方式是使用template标签进行包装，例如：

```
<template v-for="(item,index) in handle(list)">
    <div>{{index + "." + item.name}}</div>
    <div>{{item.num}}</div>
</template>
```

2.4　范例：实现待办任务列表应用

通过本章的学习，接下来尝试实现一个简单的待办任务列表应用，其可以展示当前未完成的任务项，也支持添加新的任务以及删除已经完成的任务。

2.4.1　步骤一：使用 HTML 搭建应用框架结构

使用VSCode开发工具新建一个名为todoList.html的文件。在其中编写如下HTML代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>待办任务列表</title>
    <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
    <div id="Application">
```

```
        <!-- 输入框元素，用来新建待办任务 -->
        <form @submit.prevent="addTask">
            <span>新建任务</span>
            <input
            v-model="taskText"
            placeholder="请输入任务..."
            />
            <button>添加</button>
        </form>
        <!-- 有序列表，使用v-for来构建 -->
        <ol>
            <li v-for="(item, index) in todos">
                {{item}}
                <button @click="remove(index)">
                    删除任务
                </button>
                <hr/>
            </li>
        </ol>
    </div>
</body>
</html>
```

上面的HTML代码主要在页面上定义了两块内容，表单输入框用来新建任务，有序列表用来显示当前待办的任务。运行代码，浏览器中展示的页面效果如图2-11所示。

图 2-11　待办任务应用页面布局

目前，页面中只展示了一个表单输入框，要将待办的任务添加进去，还需要实现JavaScript代码逻辑。

2.4.2　步骤二：实现待办任务列表的逻辑开发

在2.4.1节编写的代码的基础上，我们来实现JavaScript的相关逻辑。示例代码如下：

```
<script>
    const App = {
        data() {
            return {
```

```
                // 待办任务列表数据
                todos:[],
                // 当前输入的待办任务
                taskText: ""
            }
        },
        methods: {
            // 添加一条待办任务
            addTask() {
                // 判断输入框是否为空
                if (this.taskText.length == 0) {
                    alert("请输入任务")
                    return
                }
                this.todos.push(this.taskText)
                this.taskText = ""
            },
            // 删除一条待办任务
            remove(index) {
                this.todos.splice(index, 1)
            }
        }
    }
    Vue.createApp(App).mount("#Application")
</script>
```

再次运行代码，尝试在输入框中输入一些待办任务进行添加，之后可以看到，列表中已经能够将添加的任务按照添加顺序展示出来。当我们单击每一条待办任务旁边的“删除任务”按钮时，可以将当前栏目删掉，如图2-12所示。

图 2-12　待办任务应用效果

可以看到，通过Vue，我们只使用了不到30行的核心代码就完成了待办任务列表的逻辑开发。Vue在实际开发中带来的效率提升可见一斑。目前，我们的应用页面还非常简陋，并且每次刷新页面后，已经添加的待办任务会消失。如果你有兴趣，可以尝试添加一些CSS样式表来使应用的页面更加漂亮一些，通过使用前端的一些持久化功能，也可以对待办任务数据进行持久化的本地存储，这些技术后面会逐步向读者介绍。

2.5 小结与练习

本章基于Vue的模板语法介绍了Vue框架中非常重要的模板插值、模板指令等相关技术，详细介绍了如何使用Vue进行组件的条件渲染和循环渲染。本章的内容是Vue框架中的核心内容之一，仅仅使用这些技术已经可以让前端网页开发效率得到很大的提升。下面这些知识点你是否已经掌握了呢？挑战一下吧！

练习1：Vue是如何实现组件与数据间的绑定的？
温馨提示：从模板语法上分析，简述v-bind和v-model的用法与异同。

练习2：在Vue中有v-if与v-show两种条件渲染指令，它们分别怎么使用？有何异同？
温馨提示：v-if与v-show在渲染方式上有着本质的差别，从此处分析其适合的应用场景。

练习3：Vue中的模板插值应该如何使用，其是否可直接插入HTML文本？
温馨提示：需要熟练掌握v-html指令的应用。

第3章

Vue 组件的属性和方法

在定义Vue组件时，属性和方法是很重要的两部分。我们创建组件时，实现了其内部的data方法，这个方法会返回一个对象，此对象中定义的数据会存储在组件实例中，并通过响应式的更新原理来影响页面渲染。

方法定义在Vue组件的methods选项中，其与属性一样，可以在组件中访问到。本章将介绍有关Vue组件中属性与方法的相关基础知识，以及计算属性和侦听器的应用。

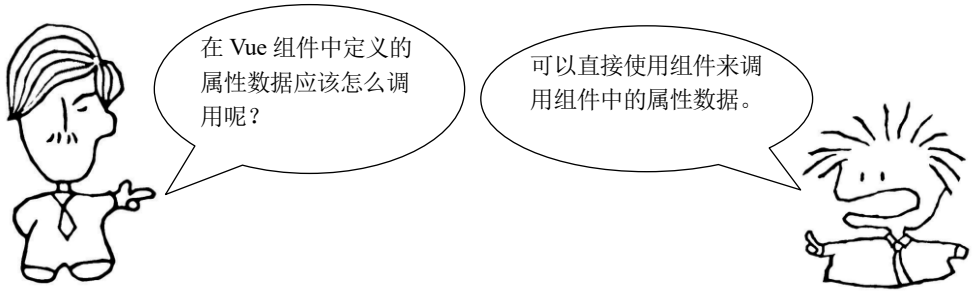
通过本章，你将学习到：

- 属性的基础知识。
- 方法的基础知识。
- 计算属性的应用。
- 侦听器的应用。
- 如何进行函数的限流。
- 表单的数据绑定技术。
- 使用 Vue 进行样式绑定。

3.1 属性与方法基础

前面我们编写Vue组件时，组件的数据都放在了data选项中，Vue组件的data选项是一个函数，组件在被创建时会调用此函数来构建响应性的数据系统。首先创建一个名为dataMethod.html的文件来编写本节的示例代码。

3.1.1 属性基础



在Vue组件中定义的属性数据，我们可以直接使用组件来调用，这是因为Vue在组织数据时，任何定义的属性都会暴露在组件中。实际上，这些属性数据是存储在组件的$data对象中的，示例如下：

```
// 定义组件
const App = {
    data() {
        return {
            count:0,
        }
    }
}
// 创建组件并获取组件实例
let instance = Vue.createApp(App).mount("#Application")
// 可以获取到组件中的data数据
console.log(instance.count)
// 可以获取到组件中的data数据
console.log(instance.$data.count)
```

运行上面的代码，通过控制台的打印可以看出，使用组件实例直接获取属性与使用$data的方式获取属性的结果是一样的，本质上访问的数据也是同一块数据，无论使用哪种方式对数据进行修改，两种方式获取到的值都会改变，示例如下:

```
// 修改属性
instance.count = 5
// 下面获取到的count的值为5
console.log(instance.count)
console.log(instance.$data.count)
```

需要注意，在实际开发中，我们也可以动态地向组件实例中添加属性，但是这种方式添加的属性不能被响应式系统跟踪，其变化无法同步到页面元素。

3.1.2 方法基础

组件的方法被定义在methods选项中，我们在实现组件的方法时，可以放心地在其中使用this关键字，Vue自动将其绑定到当前组件实例本身。例如，添加一个add方法如下：

```
methods: {
    add() {
        this.count ++
    }
}
```

我们可以将其绑定到HTML元素上，也可以直接使用组件实例来调用此方法，实例如下:

```
// 0
console.log(instance.count)
instance.add()
// 1
console.log(instance.count)
```

3.2　计算属性和侦听器

大多数情况下，我们都可以将Vue组件中定义的属性数据直接渲染到HTML元素上，但是有些场景下，属性中的数据并不适合直接渲染，需要处理后再进行渲染。在Vue中，我们通常使用计算属性或侦听器来实现这种逻辑。

3.2.1　计算属性

在前面章节的示例代码中，我们定义的属性都是存储属性。存储属性的值是我们直接定义好的，当前属性只是起到了存储这些值的作用。在Vue中，与之相对的还有计算属性，计算属性并不是用来存储数据的，而是通过一些计算逻辑来实时地维护当前属性的值。以3.1节的代码为基础，假设我们需要在组件中定义一个type属性，当组件的count属性不大于10时，type属性的值为“小”，否则type属性的值为“大”。示例代码如下：

```
// 定义组件
const App = {
    data() {
        return {
            count:0,
        }
    },
    // computed选项定义计算属性
    computed: {
        type() {
            return this.count > 10 ? "大" : "小"
        }
    },
    methods: {
    add() {
        this.count ++
    }
```

```
  }
}
// 创建组件并获取组件实例
let instance = Vue.createApp(App).mount("#Application")
// 像访问普通属性一样访问计算属性
console.log(instance.type)
```

如以上代码所示，计算属性定义在Vue组件的computed选项中，在使用时，我们可以像访问普通属性那样访问它。通常计算属性最终的值都是由存储属性通过逻辑运算得来的。计算属性强大的地方在于，当会影响其值的存储属性发生变化时，计算属性也会同步进行更新，如果有元素绑定了计算属性，其也会同步进行更新。例如，编写HTML代码如下：

```
<div id="Application">
    <div>{{type}}</div>
    <button @click="add">Add</button>
</div>
```

运行代码，单击页面上的按钮，当组件count的值超过10时，页面上对应的文案会更新成“大”。

3.2.2 使用计算属性还是函数

对于3.2.1节示例的场景，我们也可以使用函数来实现，示例代码如下：

HTML元素：

```
<div id="Application">
    <div>{{typeFunc()}}</div>
    <button @click="add">Add</button>
</div>
```

Vue组件定义：

```
const App = {
    data() {
        return {
            count:0,
        }
    },
    computed: {
        type() {
            return this.count > 10 ? "大" : "小"
        }
    },
    methods: {
        add() {
            this.count ++
        },
        typeFunc() {
            return this.count > 10 ? "大" : "小"
        }
```

```
    }
}
```

从代码的运行行为上看，使用函数与使用计算属性的结果完全一致。然而事实上，计算属性是基于其所依赖的存储属性的值的变化而重新计算的，计算完成后，其结果会被缓存，下次访问计算属性时，只要其所依赖的属性没有变化，其内的逻辑代码就不会重复执行。而函数则不同，每次访问其都会重新执行函数内的逻辑代码得到的结果。因此，在实际应用中，我们可以根据是否需要缓存这一标准来选择使用计算属性或函数。

3.2.3　计算属性的赋值

存储属性主要用于数据的存取，我们可以使用赋值运算来修改属性值。通常，计算属性只用来取值，不会用来存值，因此计算属性默认提供的是取值的方法，通常称之为get方法。但是这并不代表计算属性不支持赋值，计算属性也可以通过赋值进行存数据操作，存数据的方法我们需要手动实现，通常称之为set方法。

例如，修改上一节编写的代码中的type计算属性：

```
computed: {
    type: {
        // 实现计算属性的get方法，用来取值
        get() {
            return this.count > 10 ? "大" : "小"
        },
        // 实现计算属性的set方法，用来设置值
        set(newValue) {
            if (newValue == "大") {
                this.count = 11
            } else {
                this.count = 0
            }
        }
    }
}
```

可以直接使用组件实例计算属性type的赋值，赋值时会调用我们定义的set方法，从而实现对存储属性count的修改，示例如下：

```
let instance = Vue.createApp(App).mount("#Application")
// 初始值为0
console.log(instance.count)
// 初始状态为“小”
console.log(instance.type)
// 对计算属性进行修改
instance.type = "大"
// 打印结果为11
console.log(instance.count)
```

如以上代码所示，在实际使用中，计算属性对使用方是透明的，我们无须关心某个属性是不

是计算属性，按照普通属性的方式对其进行使用即可。但是要额外注意，如果一个计算属性只实现了get方法而没有实现set方法，则在使用时只能进行取值操作，而不能进行赋值操作。在Vue中，这类只实现了get方法的计算属性也被称为只读属性，如果对一个只读属性进行赋值操作，则会产生异常，响应的控制台会输出如下异常信息：

```
[Vue warn]: Write operation failed: computed property "type" is readonly.
```

3.2.4 属性侦听器

属性侦听是Vue非常强大的功能之一。使用属性侦听器可以方便地监听某个属性的变化，以完成复杂的业务逻辑。相信大部分使用互联网的人都使用过搜索引擎，以百度搜索引擎为例，当我们向搜索框中写入关键字后，网页上会自动关联一些推荐词供用户选择，如图3-1所示，这种场景就非常适合使用监听器来实现。

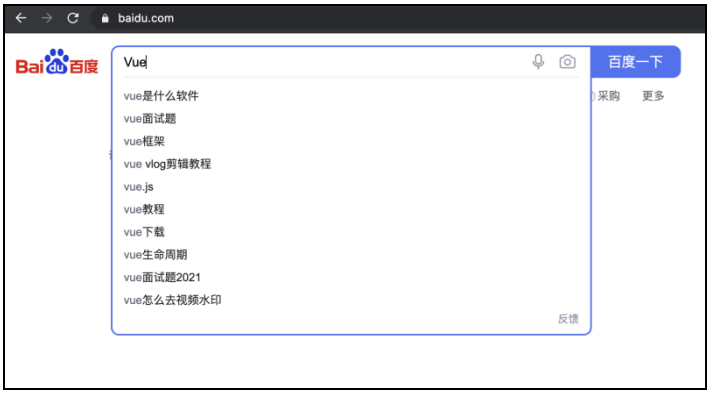


图 3-1 搜索引擎的推荐词功能

在定义Vue组件时，可以通过watch选项来定义属性侦听器。首先，创建一个名为watch.html的文件，在其中编写如下测试代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
    <div id="Application">
        <input v-model="searchText"/>
    </div>
    <script>
        const App = {
            data() {
                return {
```

```
                searchText:""
            }
        },
        watch: {
            searchText(oldValue, newValue) {
                if (newValue.length > 10) {
                    alert("文本太长了")
                }
            }
        }
    }
    Vue.createApp(App).mount("#Application")
</script>
</body>
</html>
```

运行上面的代码，尝试在页面的输入框中输入一些字符，可以看到当输入框中的字符超过10个时，就会有警告框弹出，提示输入文本过长，如图3-2所示。

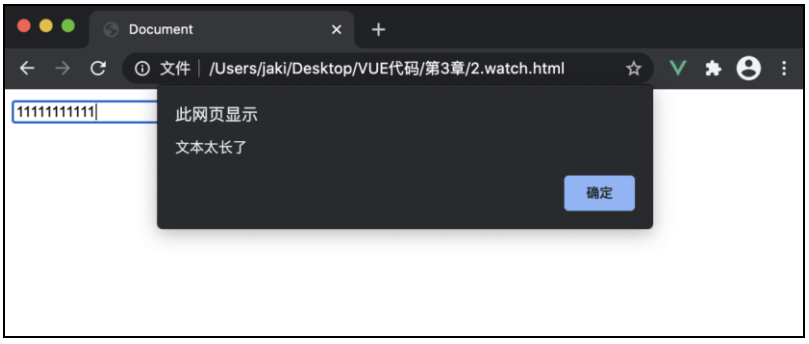


图 3-2　属性侦听器应用示例

从一些特性上看，属性侦听器和计算属性有类似的应用场景，使用计算属性的set方法也可以实现与上面的示例代码类似的功能。

3.3　进行函数限流

在工程开发中，限流是一个非常重要的概念。我们在实际开发中也经常会遇到需要进行限流的场景，例如网页上的某个按钮当用户单击后会从后端服务器进行数据的请求，在数据请求回来之前，用户额外的单击是无效的且消耗性能的。或者，网页中某个按钮会导致页面的更新，我们需要限制用户对其频繁地进行操作。这时就可以使用限流函数，常见的限流方案是根据时间间隔进行限流，即在指定的时间间隔内不允许重复执行同一函数。

本节将讨论如何在前端开发中使用限流函数。

3.3.1 手动实现一个简易的限流函数

我们先来尝试手动实现一个基于时间间隔的限流函数，要实现这样一个功能：页面中有一个按钮，单击按钮后通过打印方法在控制台输出当前的时间，要求这个按钮的两次事件触发间隔不能小于2秒。

新建一个名为throttle.html的测试文件，分析我们需要实现的功能，直接的思路是使用一个变量来控制按钮事件是否可触发，在触发按钮事件时对此变量进行修改，并使用setTimeout函数来控制2秒后将变量的值还原。使用这个思路来实现限流函数非常简单，示例代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>限流函数</title>
    <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
    <div id="Application">
        <button @click="click">按钮</button>
    </div>
    <script>
        const App = {
            data() {
                return {
                    throttle:false
                }
            },
            methods: {
                click() {
                    if (!this.throttle) {
                        console.log(Date())
                    } else {
                        return
                    }
                    this.throttle = true
                    setTimeout(() => {
                        this.throttle = false
                    }, 2000);
                }
            }
        }
        Vue.createApp(App).mount("#Application")
    </script>
</body>
</html>
```

运行上面的代码，快速单击页面上的按钮，从VSCode控制台可以看到，无论按钮被单击了多少次，打印方法都按照每2秒最多执行1次的频率进行限流。其实，在上述示例代码中，限流本身是一种通用的逻辑，打印时间才是业务逻辑，因此我们可以将限流的逻辑封装成单独的工具方法，修改核心JavaScript代码如下：

```
var throttle = false
function throttleTool(callback, timeout) {
    if (!throttle) {
        callback()
    } else {
        return
    }
    throttle = true
    setTimeout(() => {
        throttle = false
    }, timeout)
}
const App = {
    methods: {
        click() {
            throttleTool(()=>{
                console.log(Date())
            }, 2000)
        }
    }
}
Vue.createApp(App).mount("#Application")
```

再次运行代码，程序依然可以正确地运行。现在我们已经有了一个限流工具，可以为任意函数增加限流功能，并且可以任意地设置限流的时间间隔。

3.3.2　使用 Lodash 库进行函数限流

目前我们已经了解了限流函数的实现逻辑，在3.3.1节中也手动实现了一个简单的限流工具，尽管其能够满足当前的需求，细细分析，还有许多需要优化的地方。在实际开发中，每个业务函数需要的限流间隔都不同，而且需要各自独立地进行限流，我们自己编写的限流工具就无法满足了，但是得益于JavaScript生态的繁荣，有许多第三方工具库都提供了函数限流功能，它们强大且易用，Lodash库就是其中之一。

Lodash是一款高性能的JavaScript实用工具库，其提供了大量的数组、对象、字符串等边界的操作方法，使开发者可以更加简单地使用JavaScript来编程。

Lodash库中提供了debounce函数来进行方法的调用限流，要使用它，首先需要引入Lodash库，代码如下：

```
<script src="https://unpkg.com/lodash@4.17.20/lodash.min.js"></script>
```

以3.3.1节编写的代码为例，修改代码如下：

```
const App = {
    methods: {
        click: _.debounce(function(){
            console.log(Date())
        }, 2000)
    }
}
```

运行代码，体验一下Lodash限流函数的功能。

3.4 表单数据的双向绑定

双向绑定是Vue中处理用户交互的一种方式，文本输入框、多行文本输入区域、单选框与多选框等都可以进行数据的双向绑定。新建一个名为input.html的文件用来编写本节的测试代码。

3.4.1 文本输入框

文本输入框的数据绑定我们之前也使用过，使用Vue的v-model指令直接设置即可，非常简单，示例如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>表单输入</title>
    <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
    <div id="Application">
        <input v-model="textField"/>
        <p>文本输入框内容:{{textField}}</p>
    </div>
    <script>
        const App = {
            data() {
               return {
                  textField:""
                }
            }
        }
        Vue.createApp(App).mount("#Application")
    </script>
</body>
</html>
```

运行代码，当输入框中输入的文本发生变化的时候，我们可以看到段落中的文本也会同步产生变化。

3.4.2　多行文本输入区域

多行文本可以使用textarea标签来实现，textarea可以方便地定义一块区域用来显示和输入多行文本，文本支持换行，并且可以设置最多可以输入多少文本。textarea的数据绑定方式与input一样，示例代码如下：

```
<textarea v-model="textarea"></textarea>
<p style="white-space: pre-line;">多行文本内容:{{textarea}}</p>
```

在上面的代码中，为p标签设置white-space样式是为了使其可以正常地展示多行文本中的换行，运行效果如图3-3所示。

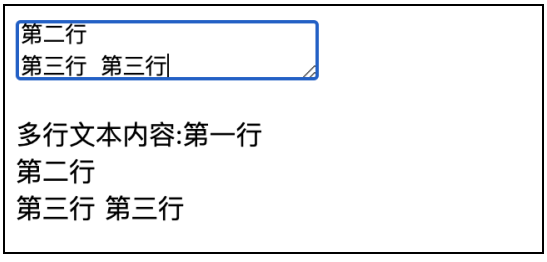


图 3-3　输入多行文本

需要注意，textarea元素只能通过v-model指令的方式来设置内容，不能直接在标签内插入文本，例如下面的代码是错误的：

```
<textarea v-model="textarea">{{text}}</textarea>
```

3.4.3　复选框与单选框

复选框为网页提供多项选择的功能，当将HTML中的input标签的类型设置为checkbox时，其就会以复选框的样式进行渲染。复选框通常成组出现，每个选项的状态只有两种：选中或未选中，如果只有一个复选框，在使用v-model指令进行数据绑定时，可以直接将其绑定为布尔值，示例如下：

```
<input type="checkbox" v-model="checkbox"/>
<p>{{checkbox}}</p>
```

运行上面的代码，当复选框的选中状态发生变化时，对应的属性checkbox的值也会切换。更多时候复选框都是成组出现的，这时我们可以为每一个复选框元素设置一个特殊的值。通过数组属性的绑定来获取每个复选框是否被选中，如果被选中，则数组中会存在其所关联的值；如果没有被选中，则数组中其关联的值会被删除掉。示例如下：

```
<input type="checkbox" value="足球" v-model="checkList"/>足球
<input type="checkbox" value="篮球" v-model="checkList"/>篮球
<input type="checkbox" value="排球" v-model="checkList"/>排球
```

```
<p>{{checkList}}</p>
```

运行代码，效果如图3-4所示。

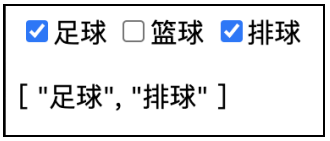


图 3-4　进行复选框数据绑定

单选框的数据绑定逻辑与复选框类似，对每一个单选框元素都可以设置一个特殊的值，并将同一组单选框绑定到同一个属性中即可，同一组中的某个单选框被选中时，对应的其绑定的变量的值也会替换为当前选中的单选框的值，示例如下：

```
<input type="radio" value="男" v-model="sex"/>男
<input type="radio" value="女" v-model="sex"/>女
<p>{{sex}}</p>
```

运行代码，效果如图3-5所示。

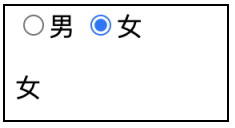


图 3-5　进行单选框数据绑定

3.4.4　选择列表

选择列表能够给用户一组选项进行选择，其可以支持单选，也可以支持多选。HTML中使用select标签来定义选择列表。如果是单选的选择列表，可以将其直接绑定到Vue组件的一个属性上，如果是支持多选的选择列表，则可以将其绑定到数组属性上。单选的选择列表示例代码如下：

```
<select v-model="select">
    <option>男</option>
    <option>女</option>
</select>
<p>{{select}}</p>
```

在select标签内部，option标签用来定义一个选项，若要使选择列表支持多选操作，则只需要为其添加上multiple属性即可，示例如下：

```
<select v-model="selectList" multiple>
    <option>足球</option>
    <option>篮球</option>
    <option>排球</option>
</select>
<p>{{selectList}}</p>
```

之后，在页面中进行选择时，按住command（control）按键即可进行多选，效果如图3-6所示。

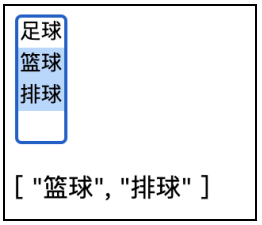


图 3-6　进行选择列表数据绑定

3.4.5　两个常用的修饰符

在对表单进行数据绑定时，我们可以使用修饰符来控制绑定指令的一些行为。比较常用的修饰符有lazy和trim。

lazy修饰符的作用有些类似于属性的懒加载。当我们使用v-model指令对文本输入框进行绑定时，每当输入框中的文本发生变化，其都会同步修改对应的属性的值。在某些业务场景下，我们并不需要实时关注输入框中文案的变化，只需要当用户输入完成后再进行数据逻辑的处理，这时就可以使用lazy修饰符，示例如下：

```
<input v-model.lazy="textField"/>
<p>文本输入框内容:{{textField}}</p>
```

运行上面的代码，只有当用户完成输入，即输入框失去焦点后，段落中才会同步到输入框中最终的文本数据。

trim修饰符的作用是将绑定的文本数据的首尾空格去掉，在很多应用场景中，用户输入的文案都是要提交到服务端进行处理的，trim修饰符处理首尾空格的特性可以为开发者提供很大的方便，示例代码如下：

```
<input v-model.trim="textField"/>
<p>文本输入框内容:{{textField}}</p>
```

3.5　样 式 绑 定

我们可以通过HTML元素的class属性、id属性或直接使用标签名来进行CSS样式的绑定，其中常用的是使用class的方式进行样式绑定。在Vue中，对class属性的数据绑定做了特殊的增强，我们可以方便地通过布尔变量控制其设置的样式是否被选用。

3.5.1　为 HTML 标签绑定 Class 属性

v-bind指令虽然可以直接对class属性进行数据绑定，但如果将绑定的值设置为一个对象，其就会产生一种新的语法规则，设置的对象中可以指定对应的class样式是否被选用。首先创建一个名为class.html的测试文件，在其中编写如下示例代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Class绑定</title>
    <script src="https://unpkg.com/vue@next"></script>
    <style>
        .red {
            color:red
        }
        .blue {
            color:blue
        }
    </style>
</head>
<body>
    <div id="Application">
        <div :class="{blue:isBlue,red:isRed}">
            示例文案
        </div>
    </div>
    <script>
        const App = {
            data() {
               return {
                   isBlue:true,
                   isRed:false,
                }
            }
        }
        Vue.createApp(App).mount("#Application")
    </script>
</body>
</html>
```

如以上代码所示，其中div元素的class属性的值会根据isBlue和isRed属性的值而改变，当只有isBlue属性的值为true时，div元素的class属性为blue；当只有isRed属性的值为true时，div元素的class属性为red。需要注意，class属性可绑定的值并不会冲突，如果设置的对象中有多个属性的值都是true，则都会被添加到class属性中。

在实际开发中，并不一定要用内联的方式为class绑定控制对象，我们也可以直接将其设置为一个Vue组件中的数据对象，修改代码如下：

HTML元素：

```
<div :class="style">
    示例文案
</div>
```

Vue组件：

```
const App = {
    data() {
       return {
           style:{
              blue:true,
              red:false
           }
        }
    }
}
```

修改后代码的运行效果与之前完全一样，更多时候我们可以将样式对象作为计算属性返回，使用这种方式进行组件样式的控制非常高效。

Vue还支持使用数组对象来控制class属性，示例如下：

HTML元素：

```
<div :class="[redClass, fontClass]">
    示例文案
</div>
```

Vue组件：

const App = {

```
    data() {
       return {
           redClass:"red",
           fontClass:"font"
        }
    }
}
```

3.5.2　绑定内联样式

内联样式是指直接通过HTML元素的style属性来设置样式，style属性可以直接通过JavaScript对象来设置样式，我们可以直接在其内部使用Vue属性，示例代码如下：

HTML元素：

```
<div :style="{color:textColor,fontSize:textFont}">
    示例文案
</div>
```

Vue组件：

```
const App = {
    data() {
       return {
           textColor:'green',
           textFont:'50px'
```

```
            }
        }
    }
```

需要注意，内联设置的CSS与外部定义的CSS有一点区别，外部定义的CSS属性在命名时，多采用“-”符号进行连接（如font-size），而内联的CSS中属性的命名采用的是驼峰命名法（如fontSize）。

内联style同样支持直接绑定对象属性，直接绑定对象属性在实际开发中更加常用，使用计算属性来承载样式对象可以十分方便地进行动态样式更新。

3.6 范例：实现一个功能完整的用户注册页面

本节尝试来完成一个功能完整的用户注册页面，并通过一些简单的 CSS 样式来使页面布局得漂亮一些。

3.6.1 步骤一：搭建用户注册页面

我们计划搭建一个用户注册页面，页面由标题、一些信息输入框、偏好设置和确认按钮这几个部分组成。首先，创建一个名为register.html的测试文件，按照常规的开发习惯，我们先来搭建HTML框架结构，编写代码如下：

```
    <div class="container" id="Application">
        <div class="container">
            <div class="subTitle">加入我们，一起创造美好世界</div>
            <h1 class="title">创建你的账号</h1>
            <div v-for="(item, index) in fields" class="inputContainer">
                <div class="field">{{item.title}} <span v-if="item.required"
style="color: red;">*</span></div>
                <input class="input" :type="item.type" />
                <div class="tip" v-if="index == 2">请确认密码程度需要大于6位</div>
            </div>
            <div class="subContainer">
                <div class="setting">偏好设置</div>
                <input class="checkbox" type="checkbox" /><label class="label">接
收更新邮件</label>
            </div>
            <button class="btn">创建账号</button>
        </div>
    </div>
```

上面的代码提供了主页页面所需要的所有元素，并且为元素指定了class属性，同时也集成了一些Vue的逻辑，例如循环渲染和条件渲染。下面定义Vue组件，示例代码如下：

```
    const App = {
        data() {
```

```
        return {
            fields:[
                {
                    title:"用户名",
                    required:true,
                    type:"text"
                },{
                    title:"邮箱地址",
                    required:false,
                    type:"text"
                },{
                    title:"密码",
                    required:true,
                    type:"password"
                }
            ],
        }
    }
}
Vue.createApp(App).mount("#Application")
```

上面的代码定义了Vue组件中与页面布局相关的一些属性，截至目前我们还没有处理与用户交互相关的逻辑，先将页面元素的CSS样式补齐，示例代码如下：

```
<style>
    .container {
        margin:0 auto;
        margin-top: 70px;
        text-align: center;
        width: 300px;
    }
    .subTitle {
        color:gray;
        font-size: 14px;
    }
    .title {
        font-size: 45px;
    }
    .input {
        width: 90%;
    }
    .inputContainer {
        text-align: left;
        margin-bottom: 20px;
    }
    .subContainer {
        text-align: left;
    }
    .field {
        font-size: 14px;
    }
```

```
        .input {
            border-radius: 6px;
            height: 25px;
            margin-top: 10px;
            border-color: silver;
            border-style: solid;
            background-color: cornsilk;
        }
        .tip {
            margin-top: 5px;
            font-size: 12px;
            color: gray;
        }
        .setting {
            font-size: 9px;
            color: black;
        }
        .label {
            font-size: 12px;
            margin-left: 5px;
            height: 20px;
            vertical-align:middle;
        }
        .checkbox {
            height: 20px;
            vertical-align:middle;
        }
        .btn {
            border-radius: 10px;
            height: 40px;
            width: 300px;
            margin-top: 30px;
            background-color: deepskyblue;
            border-color: blue;
            color: white;
        }
    </style>
```

运行代码，页面效果如图3-7所示。

在注册页面中，元素的UI效果预示了其部分功能，例如在输入框上方有些标了红星，表示此项是必填项，即如果用户不填写，将无法完成注册操作。对于密码输入框，我们将其类型设置为password，当用户在输入文本时，此项会被自动加密。下一节将重点对页面的用户交互逻辑进行处理。

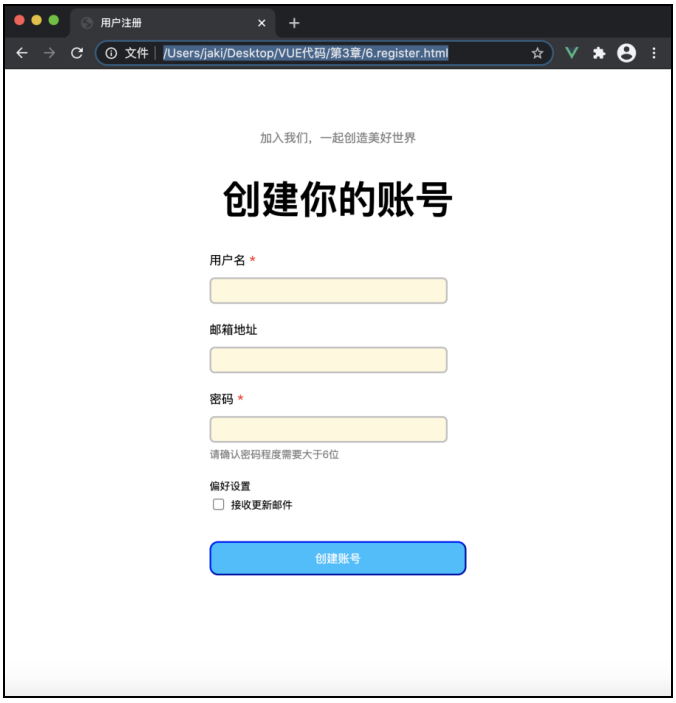


图 3-7　简洁的用户注册页面

3.6.2　步骤二：实现注册页面的用户交互

以我们编写好的注册页面为基础，本节来为其添加用户交互逻辑。在用户单击注册按钮时，我们需要获取用户输入的用户名、密码、邮箱和偏好设置，其中用户名和密码是必填项，并且密码的长度需要大于6位，对于用户输入的邮箱，也可以使用正则来对其进行校验，只有格式正确的邮箱才允许被注册。

由于页面中的3个文本输入框是通过循环动态渲染的，因此在对其进行绑定时，我们也需要采用动态的方式进行绑定。首先在HTML元素中将需要绑定的变量设置好，示例如下：

```
    <div class="container" id="Application">
        <div class="container">
            <div class="subTitle">加入我们，一起创造美好世界</div>
            <h1 class="title">创建你的账号</h1>
            <div v-for="(item, index) in fields" class="inputContainer">
                <div class="field">{{item.title}} <span v-if="item.required"
style="color: red;">*</span></div>
                <input v-model="item.model" class="input" :type="item.type" />
                <div class="tip" v-if="index == 2">请确认密码程度需要大于6位</div>
            </div>
            <div class="subContainer">
                <div class="setting">偏好设置</div>
                <input v-model="receiveMsg" class="checkbox" type="checkbox"
/><label class="label">接收更新邮件</label>
            </div>
```

```
        <button @click="createAccount" class="btn">创建账号</button>
    </div>
</div>
```

完善Vue组件如下：

```
const App = {
    data() {
        return {
            fields:[
                {
                    title:"用户名",required:true,type:"text",
                    model:""
                },{
                    title:"邮箱地址",required:false,type:"text",
                    model:""
                },{
                    title:"密码",required:true,type:"password",
                    model:""
                }
            ],
            receiveMsg:false
        }
    },
    computed:{
        name: {
            get() {
                return this.fields[0].model
            },
            set(value){
                this.fields[0].model = value
            }
        },
        email: {
            get() {
                return this.fields[1].model
            },
            set(value){
                this.fields[1].model = value
            }
        },
        password: {
            get() {
                return this.fields[2].model
            },
            set(value){
                this.fields[2].model = value
            }
        }
    },
    methods:{
```

```
            emailCheck() {
                var verify =
/^\w[-\w.+]*@([A-Za-z0-9][-A-Za-z0-9]+\.)+[A-Za-z]{2,14}/;
                if (!verify.test(this.email)) {
                    return false
                } else {
                    return true
                }
            },
            createAccount() {
                if (this.name.length == 0) {
                    alert("请输入用户名")
                    return
                } else if (this.password.length <= 6) {
                    alert("密码设置需要大于6位字符")
                    return
                } else if (this.email.length > 0 && !this.emailCheck(this.email))
{
                    alert("请输入正确的邮箱")
                    return
                }
                alert("注册成功")

console.log(`name:${this.name}\npassword:${this.password}\nemail:${this.email}
\nreceiveMsg:${this.receiveMsg}`)
            }
        }
    }
    Vue.createApp(App).mount("#Application")
```

上面的代码通过配置输入框field对象来实现动态数据绑定，为了方便值的操作，我们使用计算属性对几个常用的输入框数据实现了便捷的存取方法，这些技巧都是本章介绍的核心内容。当用户单击“创建账号”按钮时，createAccount方法会进行一些有效性校验，我们对每个字段需要满足的条件依次进行校验即可，上面的示例代码使用了正则表达式对邮箱地址的有效性进行检查。

现在运行代码，在浏览器中尝试进行用户注册的操作。截至目前，我们完成了一个较为完善的客户端的注册页面，在实际应用中，最终的注册操作还需要与后端进行交互。

3.7　小结与练习

本章介绍了Vue组件中有关属性和方法的基础应用，并且通过一个较为完整的范例练习了数据绑定、循环与条件渲染以及计算属性相关的核心知识。相信通过本章的学习，读者对Vue的使用可以有了更深的理解。

练习1：Vue中的计算属性和普通属性有什么区别？

温馨提示：普通属性的本质是存储属性，计算属性的本质是调用函数。从此方面思考其异同，

并且思考它们各自适用的场景。

练习2：属性侦听器的作用是什么？

温馨提示：当数据变化会触发其他相关的业务逻辑时，可以尝试使用属性监听器来实现。

练习3：你能够手动实现一个限流函数吗？

温馨提示：结合本章中的示例思考实现限流函数的核心思路。